Software Maintenance and Support Contract — Final Report

Tom Gee
Covenant Systems Development

Covenant Systems Development
27 Bray St, Beeton, Ontario, L0G 1A0

Project Manager: Bill Fraser 416-635-2064

Contract Number: W7711-036681/001/TOR

Contract Scientific Authority: Bill Fraser 416-635-2064

Terms of release: The information contained herein is proprietary to Her Majesty and is provided to the recipient on the understanding that it will be used for information and evaluation purposes only. Any commercial use including use for manufacture is prohibited. Release to third parties of this publication or information contained herein is prohibited without the prior written consent of Defence R&D Canada.

Defence Research & Development - Toronto

Contractor Report

September 2004

Author

Tom Gee

Approved by

Bill Fraser
Principal Investigator

Approved for release by

Bill Fraser
Project Manager

The scientific or technical validity of this Contractor Report is entirely the responsibility of the contractor and the contents do not necessarily have the approval or endorsement of Defence R&D Canada.

© Her Majesty the Queen as represented by the Minister of National Defence, 2004

© Sa majesté la reine, représentée par le ministre de la Défense nationale, 2004

106-03158/

Table of contents

Table of contents i

List of figures i

Abstract iii

Résumé iii

Executive summary v

Sommaire vi

1 Goals 1

2 Progress Accomplished 2

2.1 Establishing the External Software Platform 2

2.2 Enhance GUARDIAN Architecture and Support 4

2.2.1 Configuring the Processing Network 6

2.2.2 Navigating the Data Repository 6

2.2.3 Monitoring via a Graphical Waveform User Interface . 10

2.2.4 Optimizing CPU and Storage Resources 13

3 Future Directions 14

Annexes 15

A C++ Library Code 15

B Configuration Script 77

References 96

Index 98

List of figures

1 Sample OpenDX display window 3

2 Sample PDP++ model view 3
3 Sample dicom2-converted MRI image (inverted) 4
4 Sample dicom2-converted extracted text information 5
5 Sample processing network 7
6 Sample *datatrac* database window 8
7 Sample konqueror window as file manager 9
8 Sample nautilus window 10
9 Original GUARDIAN user interface 11
10 Original GUARDIAN user interface with graphing 11
11 Sample paraGUI interface 12
12 Sample graphical representation of heart rate. 13

Abstract

To provide an effective platform for acquiring and processing a large number of high-speed channels, we were engaged to explore several third party software packages. We were also tasked with augmenting the existing software architecture, the GUARDIAN system.

We examined packages for real-time modeling, display, storage and importing outside data. We evaluated the tasks required to configure and apply GUARDIAN to this type of problem, as well as those required to navigate the stored data after acquiring. For those tasks which were onerous, we explored ways of simplifying them. We examined various options for creating a graphical configuration editor, a filesystem-type interface to the storage database, and a real-time waveform-oriented user interface, with the latter proceeding through the design and initial implementation phases.

The existing system, combined with the third party packages, is a suitable platform for this research, if somewhat awkward in its employ. By completing the modest projects started under this program, the ease and efficiency of the system will be increased markedly.

Résumé

Afin de fournir une plate-forme efficace pour acquérir et traiter un grand nombre de canaux rapides, on nous a engagés étudier fond plusieurs progiciels tiers. On nous a également confié la mission de renforcer l'architecture du logiciel existant, soit le système GUARDIAN.

Nous avons examiné des progiciels pour en analyser la modélisation, l'affichage, le stockage et l'importation en temps réel de données extérieures. Nous avons évalué les tâches requises pour configurer et appliquer le système GUARDIAN à ce genre de problème, d'une part, et celles pour explorer les données stockées après acquisition, d'autre part. Nous avons cherché des moyens pour simplifier les tâches qui se sont avérées exigeantes. Nous avons étudié différentes options pour créer un éditeur de configuration graphique, une interface du genre système de fichier à la base de données de stockage et une interface utilisateur en temps réel focalisé sur la forme d'onde. L'interface utilisateur a traversé les phases de conception et de mise en oeuvre.

Combiné aux progiciels tiers, le système existant représente une plate-forme conve-

nable pour la recherche, bien qu'il soit plutôt peu maniable. Les modestes projets accomplis dans le cadre de ce programme augmenteront notablement la maniabilité et l'efficacité du système.

Executive summary

DRDC Toronto is preparing for a research thrust requiring both physiological modeling and physical experimentation. The focus of this contract has been to prepare a software platform for serving these ends.

A sophisticated data acquisition, processing and storage architecture called GUARDIAN has already been developed within DRDC. Our task was to augment the system to support high-speed acquisition and processing over a large number of channels, to enhance existing software as well as employing third party software to process the data in real-time and offline, as well as to assist DRDC personnel in integrating application-specific modeling code with the rest of the system.

We identified and deployed several third party packages for real-time modeling, display, storage and importing of outside data. We also ensured that GUARDIAN was capable of acquiring and processing the predicted data load and identified several issues regarding the complexity of configuring the system and navigating the resulting data.

For those issues identified, we proposed and designed a set of solutions. To simplify configuring the system, we proposed and scoped a graphical configuration editor. To ensure reliable use of GUARDIAN with high-speed channels, we designed and began implementation of a waveform-oriented graphical user interface which provides a highly customizable presentation with real-time display. Finally to simplify navigation of the acquired and processed data, we propose and scope a more intuitive and flexible interface to the existing database system.

The results of these modest projects, carried to completion, will be a platform well-suited to this ambitious application. The user will find the system capable, flexible and intuitive in the support of any number of paths of inquiry.

Tom Gee; 2004; Software Maintenance and Support Contract — Final Report; ; Defence Research & Development - Toronto.

Sommaire

RDDC Toronto se prépare pour un vecteur de recherche qui exige à la fois une simulation physiologique et une expérimentation physique. Le présent contrat se focalise sur l'élaboration d'une plate-forme logicielle dans le but de satisfaire à ces besoins.

On a déjà réalisé au sein de RDDC une architecture sophistiquée d'acquisition, de traitement et de stockage de données appelée GUARDIAN. Notre tâche a été d'augmenter le système afin de faciliter l'acquisition et le traitement sur un grand nombre de canaux. On devait aussi renforcer le logiciel existant, employer un logiciel tiers pour traiter les données en temps réel et hors ligne ainsi qu'à aider le personnel de RDDC à intégrer le code de modélisation particulier à l'application au reste du système.

Nous avons désigné et déployé plusieurs progiciels tiers pour modéliser, afficher, stocker et importer en temps réel des données de l'extérieur. Nous avons veillé également à ce que le GUARDIAN puisse acquérir et traiter un chargement de données prévu et nous avons cerné plusieurs problèmes dus à la complexité qu'il y a à configurer le système et à explorer les données résultantes.

En ce qui concerne les problèmes cernés, nous avons proposé et élaboré des solutions. Pour simplifier la configuration du système, nous avons proposé et spécifié un éditeur de configuration graphique. Afin que le GUARDIAN puisse être utilisé de façon fiable avec des canaux à haute vitesse, nous avons élaboré et commencé à mettre en oeuvre une interface utilisateur graphique focalisé sur la forme d'onde, qui fournit une présentation personnalisable avec affichage en temps réel. Enfin, pour simplifier la navigation des données acquises et traitées, nous proposons et spécifions une interface plus souple et plus intuitive pour le système de base de données existant.

Accomplis, ces modestes projets auront comme résultats une plate-forme qui convient très bien à cette application ambitieuse. L'utilisateur trouvera le système capable, souple et intuitif quel que soit le nombre de chemins d'accès pour interroger.

Tom Gee; 2004; Software Maintenance and Support Contract — Final Report; ; Research & Development pour la Defense du Canada - Toronto.

1 Goals

As shown in the statement of work [1], this contract involved several interrelated tasks:

> The contractor will be required to *(1)* undertake the modification and integration of real-time and off-line modeling and analysis software into the Linux based IMEOS/Guardian system architecture, *(2)* modify and test the Guardian architecture for 200+ data channel capacity, and *(3)* assist DRDC personnel in integrating and modifyiong [sic] the C++ code provided by the PDP++ modeling software. [Numbers added]

We will briefly examine how these goals developed during the course of the contract.

1. *Undertake the modification and integration of real-time and off-line modeling and analysis software in the Linux-based IMEOS/Guardian system architecture*

   The focus developed under this task was to install and become familiar with the offline analysis, storage and display technologies of OpenDX [2], HDF5 [3], dicom2 [4] and the DICOM Central Test Node (CTN) [5] software. DICOM and OpenDX were particularly interesting as we were exploring the idea of projecting EEG and EMG data onto MRI and ultrasound images of the subject's anatomy. As well, the GUARDIAN software architecture [6] developed under a previous contract [7] required some modification to efficiently support multiple channel EEG; modifications that included creating a basic real-time user interface.

2. *Modify and test the Guardian architecture for 200+ data channel capacity*

   Designed as a highly-scalable architecture, GUARDIAN was already theoretically capable of 200+ channels. However, the difficulty of configuring these channels using the GUARDIAN data path network paradigm was daunting, as was the task of navigating the acquired data stored offline. Accordingly, the issues we addressed included designing a user interface to the real-time system, surveying various technologies for creating a configuration tool, and designing a simplified paradigm for addressing the resulting data stored in the normal destination for GUARDIAN, the *datatrac* database [8].

3. *Assist DRDC personnel in integrating and modifying the C++ code provided by the PDP++ modeling software*

   This task was ultimately restricted to installing PDP++ [9] and gaining a small degree of familiarity with it. DRDC did not ultimately produce any PDP++

or C++ code in the timespan of this short contract. However, the modular and generic design of GUARDIAN should make such integration straightforward.

2 Progress Accomplished

Although the contract identified three tasks, the vestigial results of the third combined with the overlap between the first two make it simpler to examine the progress made as two main categories, namely, the establishing of the external software platform and the enhancing of the GUARDIAN architecture and support software.

2.1 Establishing the External Software Platform

Using a number of packages external to GUARDIAN, we began laying a foundation of several real-time and offline storage, analysis and display technologies. We investigated, identified, installed and configured the following:

OpenDX is a superb, open-source version of IBM's original Data Explorer. This program takes multidimensional data from various sources and allows it to be represented in a myriad of complex and ingenious ways, such as that shown in figure 1.

HDF5 is version 5 of the NCSA's Hierarchical Data Format, a file format and library for storing a hierarchical, mixed type data. Although we do not yet use HDF, it is one of the main formats supported by OpenDX and under consideration by DRDC Toronto.

CTN is a collection of software programs used to create, manipulate and export medical data such as MRI images. We need access to such information to create accurate models for the display and interpretation of physiological data.

dicom2 is a DICOM image exporter which provides a simpler interface than that of the CTN for exporting medical images.

PDP++ is a simulation environment for developing and testing connectionist models of neural function. A sample image is shown in figure 2 taken from the user guide [10].

We spent considerable time gaining a basic knowledge of these packages, particularly OpenDX. Since this program is capable of receiving streaming data, we explored the idea of using it as a user interface during real-time acquisition. The idea

Figure 1: *Sample OpenDX display window*

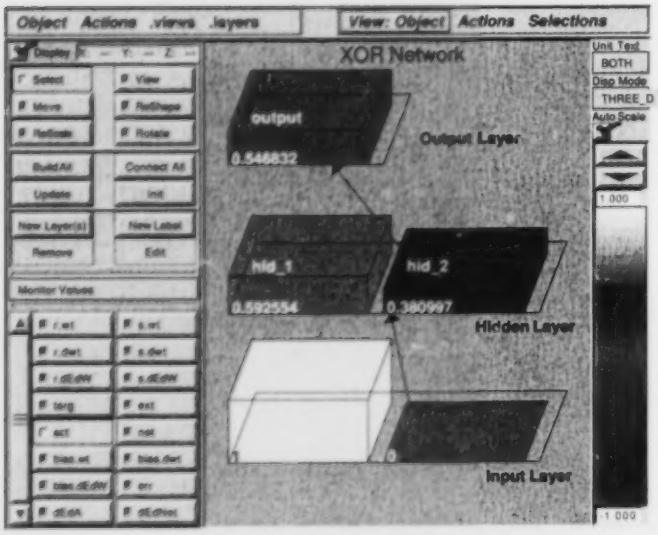

Figure 2: *Sample PDP++ model view*

Figure 1: *Sample OpenDX display window*

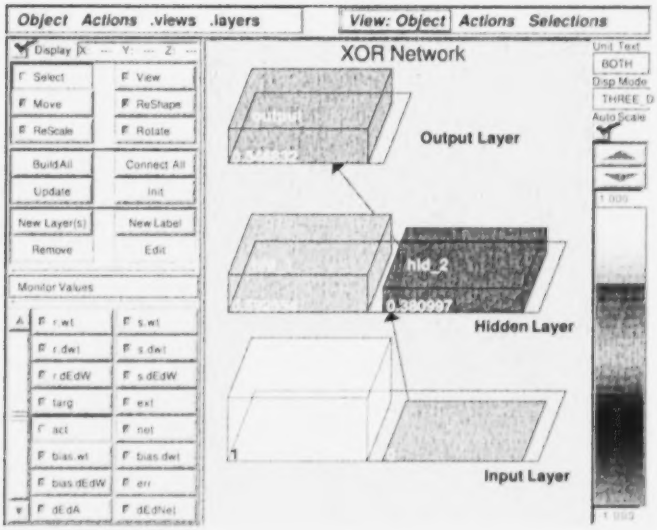

Figure 2: *Sample PDP++ model view*

is compelling and reasonable, but at the moment the need for such a display does not justify the effort required.

We also used the dicom2 program and some sample images that came with CTN. It has proven adequate to export the data both as a JPEG graphic (figure 3) and as text information (figure 4) giving dimensions, scan positions, etc. We were unable to locate a sample of MRI scans that were suitable for volumetric rendering. We will likely need to wait until subject MRI scans are available before making much progress in this area.

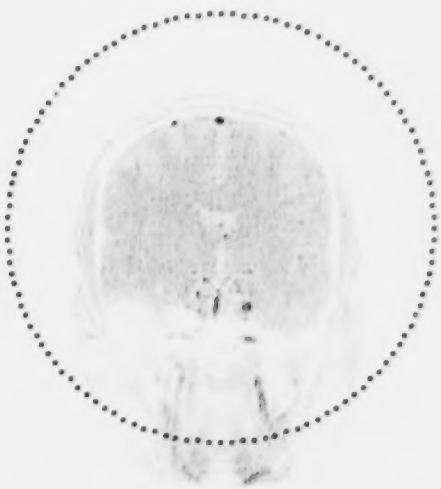

Figure 3: *Sample dicom2-converted MRI image (inverted)*

2.2 Enhance GUARDIAN Architecture and Support

To enhance GUARDIAN for the established tasks required work on several issues, specifically:

1. configuring high-density processing networks
2. navigating a high-density stored data repository
3. confirming and monitoring data quality via a graphical waveform-oriented user interface
4. optimizing of CPU and onboard storage resources

0002 0001	2	META File Meta Information Version	OB or OW
0002 0003	48	META Media Stored SOP Instance UID	1.3.12.2.1107.5.8.1.12345.20010530134210056278O
0002 0010	20	META Transfer Syntax UID	1.2.840.10008.1.2.1
0008 0008	22	ID Image Type	ORIGINAL\PRIMARY\OTHER
0008 0018	48	ID SOP Instance UID	1.3.12.2.1107.5.8.1.12345.20010530134210056278O
0008 0020	8	ID Study Date	20010529
0008 0021	8	ID Series Date	20010529
0008 0022	8	ID Acquisition Date	20010529
0008 0023	8	ID Image Date	20010529
0008 0030	14	ID Study Time	150951.075000
0008 0032	14	ID Acquisition Time	155752.451000
0008 0060	2	ID Modality	MR
0008 0070	8	ID Manufacturer	SIEMENS
0008 0080	10	ID Institution Name	Princeton
0008 1010	8	ID Station Name	allegra
0008 103e	10	ID Series Description	NYU/Scout
0008 1090	16	ID Manufacturer Model Name	MAGNETOM Allegra
0009 0012	20		SIEMENS CM VA0 CMS
0009 1041	8		SIEMENS
0009 1210	8		STANDARD
0010 0010	28	PAT Patient Name	kw29may2001_brain_stability
0010 0020	8	PAT Patient ID	wicklow
0010 0030	8	PAT Patient Birthdate	19620122
0010 0040	2	PAT Patient Sex	M
0010 1010	4	PAT Patient Age	039Y
0010 1030	6	PAT Patient Weight	70
0011 0010	14		SPI RELEASE 1
0011 0011	20		SIEMENS CM VA0 CMS
0018 0015	4	ACQ Body Part Examined	HEAD
0018 0020	2	ACQ Scanning Sequence	RM
0018 0021	4	ACQ Sequence Variant	OSP
0018	2	ACQ MR Acquisition	2D

Figure 4: *Sample dicom2-converted extracted text information*

To reduce the complexity of this report, we will examine each issue in turn.

2.2.1 Configuring the Processing Network

As noted, GUARDIAN was designed with scalability in mind. The requirement of 200+ input data channels required no modification to the architectural design or prototype implementation [11]. However, the expressive power of GUARDIAN does make initial configuration more difficult than other, simpler systems used previously. In particular, the concept of "virtual data paths" — which allow the environment of the system to be modeled along with the internal elements of the software — place considerable demands on the user.

A processing network consists of an aggregate of interconnected transformations. A transformation is an entity with a collection of data sinks and a collection of data sources. Various sinks are connected to various sources to form a network. Typically, a script file is used to describe the elements of the processing network, including transformations, source and sink flows, and connections. For example, figure 5 shows a basic anti-G valve configuration, using 9 transforms, 26 flows and 14 connections (dotted lines).

A network configured for 200 data channels will contain at a minimum, 2 transformations, 600 flows and 200 connections. Although not complex to create, such a configuration script represents a large amount of typing and is an onerous task to create manually, as the author knows all too well.

Accordingly, we looked at various technologies for simplifying this task. We were primarily concerned with software packages or libraries that supported the creation of graphs, that is, collections of nodes and arcs. A graph is the most intuitive way to display a processing network, although each node and arc must be able to have several parameters applied to them.

We found several such packages but none were suitable. The requirement to apply parameters to nodes and arcs exceed anything we have found available. We do use the *dot* program from the graphviz package [12] to generate simple network representations (such as figure 5), but it is not adequate to serve as a graph editor. Accordingly, we have started creating a script editor entirely from scratch, with the visual design borrowing from pgaccess [13] and Khoros' Canata.

2.2.2 Navigating the Data Repository

Another challenge arising from the large number of acquired channels is that navigating the results. Currently, GUARDIAN data is imported into a *datatrac* database, from where the results can be exported in a number of different formats via the

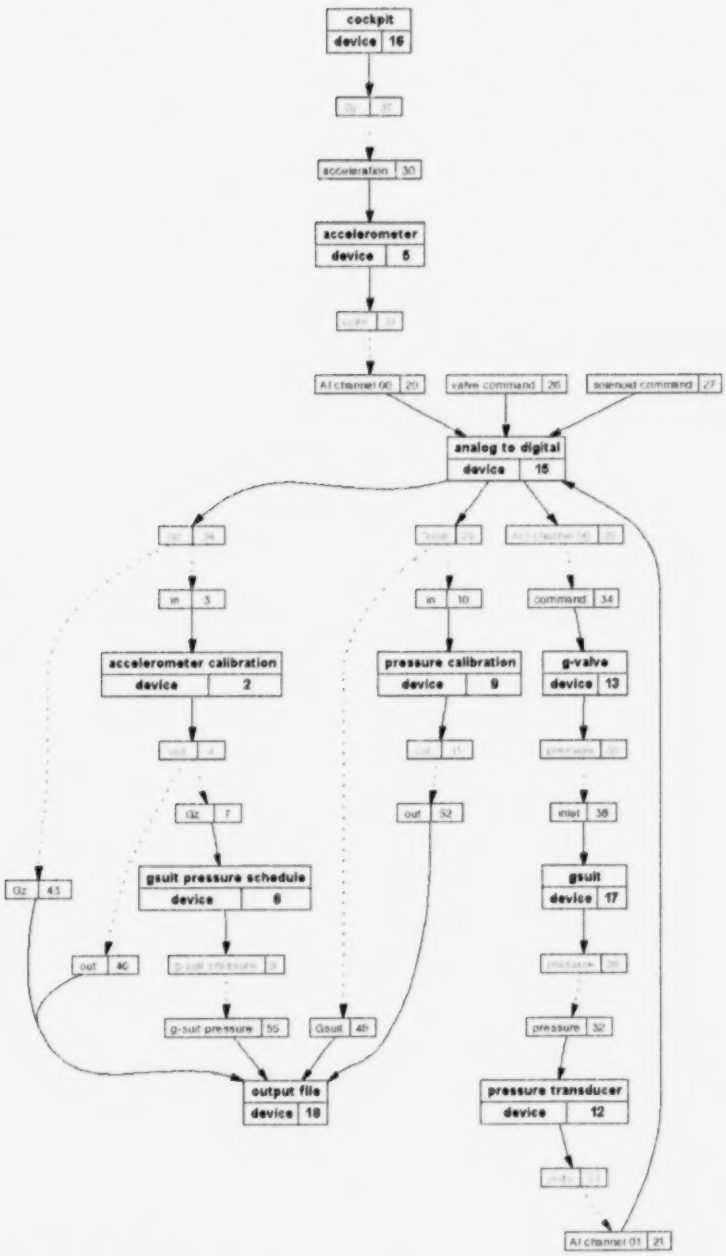

Figure 5: *Sample processing network*

web interface (figure 6). However, navigating hundreds of data channels through this interface is difficult. Accordingly, we began to examine different options for providing more intuitive and customizable access by the user.

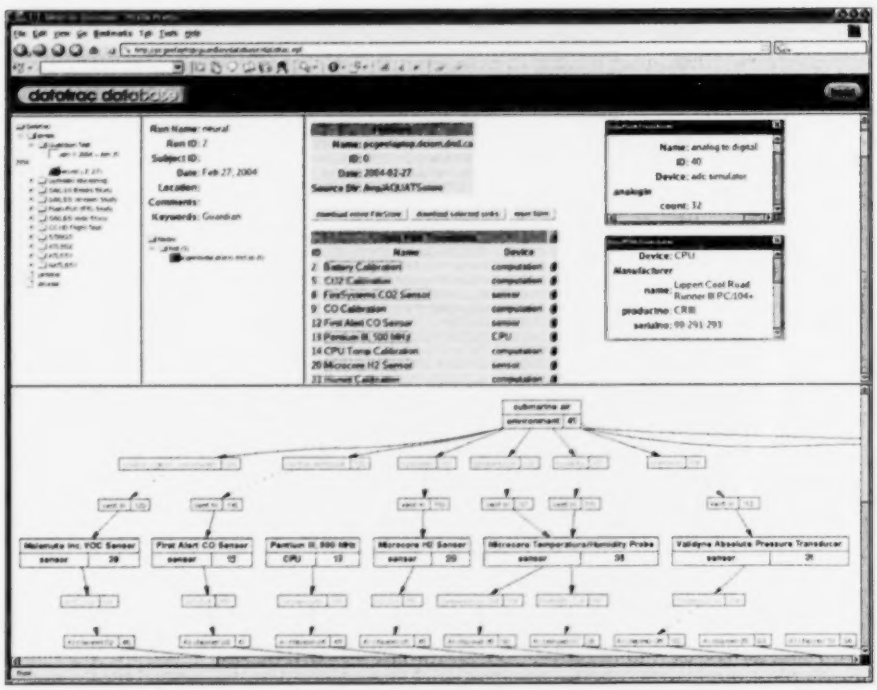

***Figure 6:** Sample* datatrac *database window*

The most widely used and arguably successful paradigms for data organization is the file system: a collection of files and directories arranged in a hierarchical and recursive manner. As this interface is very familiar and intuitive to all computer users, we decided to implement a file system interface to the *datatrac* database.

There were three main technologies which we examined for this purpose. The excellent konqueror [14] file manager and web browser provided by the KDE [15] project seemed to be good choice. This file manager (figure 7) is highly customizable and pleasant to work with. It uses the KDE KIO [16] connection technology which seemed to be a good fit to our requirements. However, KIO is layered on top of the Qt library [17] which is licensed via the GNU General Public License (GPL) [18] which would encumber our software unduly.

The gnomeVFS [19] virtual file system library was another possible option. Using this package would require the researcher to use Gnome's *nautilus* file manager (figure 8), but this manager is very similar to that of Microsoft Windows, with which

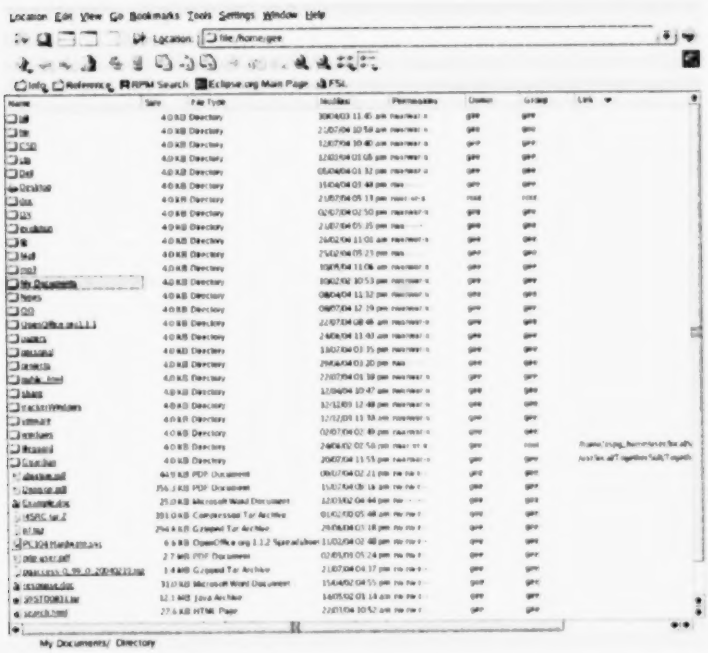

Figure 7: *Sample konqueror window as file manager*

most users are familiar. gnomeVFS is also a simpler if more restricted technology than KIO. However, this library is also covered under GPL and so is unsuitable for integration at this low level.

Perhaps a word about GPL would be appropriate here. At DRDC Toronto and at Covenant Systems Development, we make extensive use of Open Source and GPL software packages. However, when looking to deliver an integrated software library, if that library would require GPL'd source for basic operation, we typically find another option. Although the GPL represents a noble social experiment and a laudable ideal, Covenant Systems will rarely accept the encumbrance it requires on any software developed for an employer where the intellectual property and source reside with the employer.

Finally, we discovered the excellent "Filesystem in User Space" or FUSE package [20]. This library is covered by the LGPL [21] which does not involve encumbering the software developed under contract. Also, it provides a full, low-level filesystem interface, which allows the user to employ any file manager they desire, including konqueror and nautilus. FUSE also avoids much of the complexity of KIO while being more expressive than gnomeVFS. This is the package we have decided to move forward with in future development.

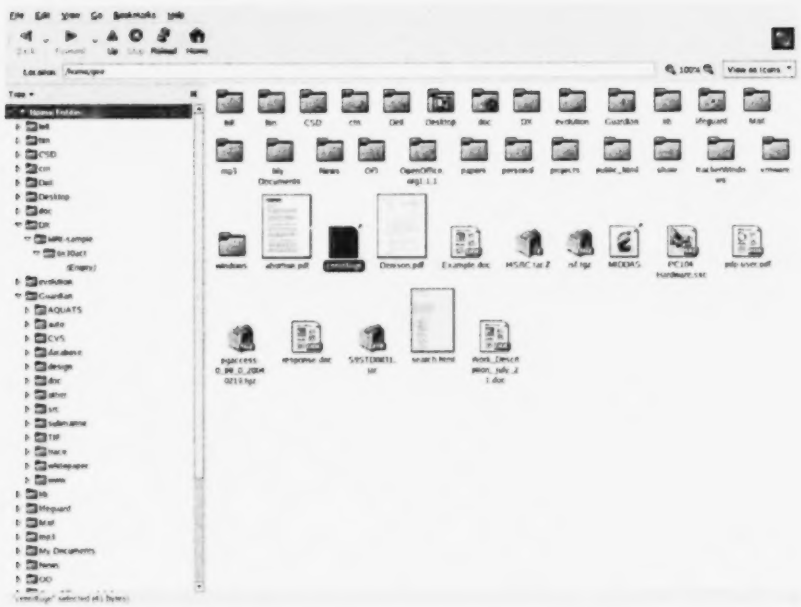

Figure 8: *Sample nautilus window*

2.2.3 Monitoring via a Graphical Waveform User Interface

The existing user interface available within GUARDIAN was originally designed as a display for monitoring atmosphere quality in military vehicles [22]. Such parameters must be displayed with great accuracy, requiring numeric displays (figure 9), but change slowly, meaning that any waveform representation can be updated at a low rate (figure 10). These requirements are completely inverted for this new application of GUARDIAN, where values such as EEG and EMG are dimensionless and their magnitude almost meaningless, but the waveform shape and frequency composition of these high-speed channels is all important.

Accordingly, we have undertaken the design and implementation of a generic waveform plotting graphical display. Within GUARDIAN, the design of such a display has several challenges, the greatest two being:

1. representing multidimensional data in an intuitive format, and

2. representing the content of the data when a simple label does not suffice as the data is the sum total of the preceding processing network.

We designed a new active object [23] class called `WaveUI`. Using the paraGUI [24] graphical toolkit overlayed on the multiplatform SDL library [25] and the OpenGL

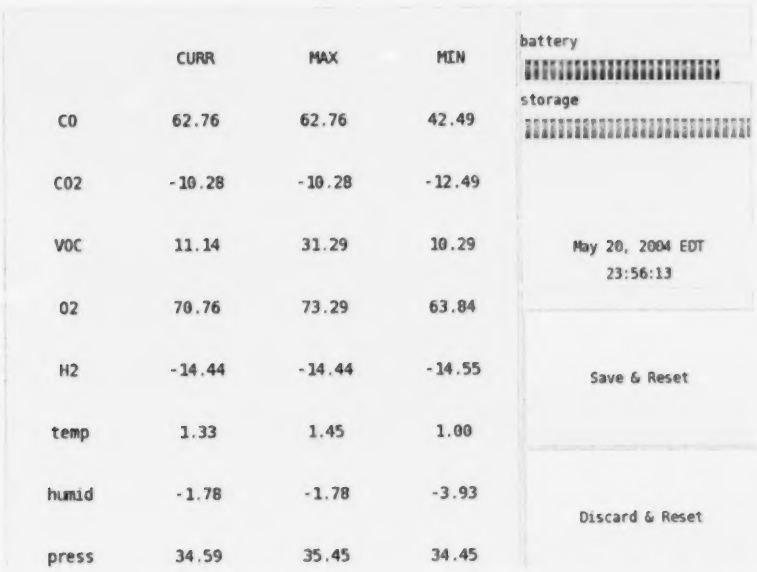

Figure 9: *Original* GUARDIAN *user interface*

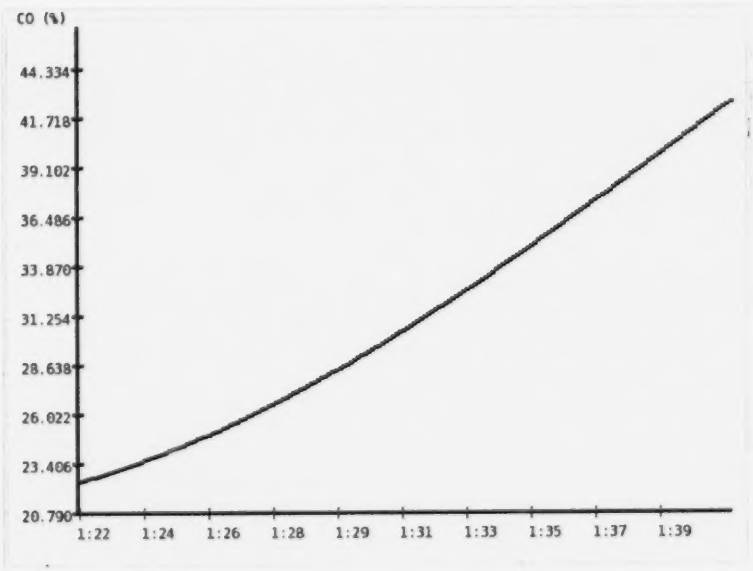

Figure 10: *Original* GUARDIAN *user interface with graphing*

accelerated graphics library [26], we gained the ability to create a full-featured interface with the typical GUI items — pull-down menus, check boxes, buttons — combined with the powerful, accelerated graphics rendering provided by modern video cards. Figure 11 shows an extensive if overenthusiastic example.

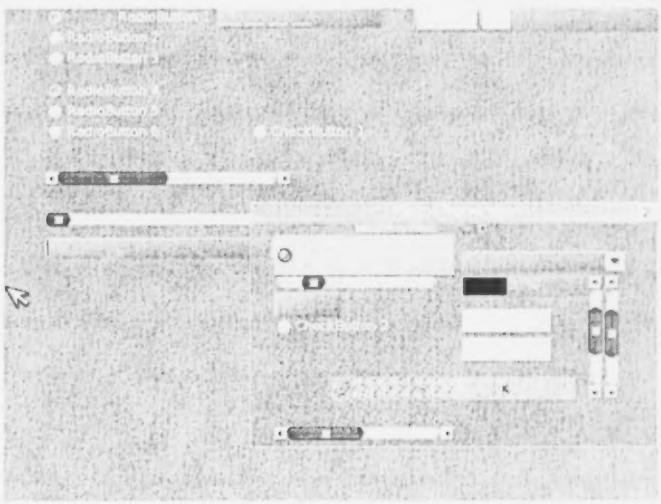

Figure 11: *Sample paraGUI interface*

The `WaveUI` class aggregates any number of lower-level graphing objects which themselves handle user interaction through controls, data representation through graphical plots and data representation through digital displays. Currently only the graphical plot classes have been implemented.

Each graphical plot object — the `GLGraph` class — can represent an arbitrary number of individual traces, encapsulated in the `GraphTrace` class. Each `GraphTrace` defines the mapping of a data flow's dimensions to various graphical elements, scales the data both in X and Y dimensions, and handles colour mapping.

Through these mechanisms, the user can create displays that represent multidimensional data in flexible and hopefully intuitive ways. For example, suppose one of the data flows we wish to display is heart rate, which consists of a stream of data triplets: timestamp, heart rate value, reliability index. The user may elect to graph 30 seconds of this data on the screen, using a scrolling display where the first five seconds of data use 25% of the screen area and the remaining 25 seconds is squeezed into the other 75%. This would provide a magnified display of the most recent data with a more condensed display of older data, keeping trends visible without details. The user may choose to map the heart rate value dimension as the Y value and the reliability index value to the trace's alpha. Thus, as the reliability

accelerated graphics library [26], we gained the ability to create a full-featured interface with the typical GUI items — pull-down menus, check boxes, buttons — combined with the powerful, accelerated graphics rendering provided by modern video cards. Figure 11 shows an extensive if overenthusiastic example.

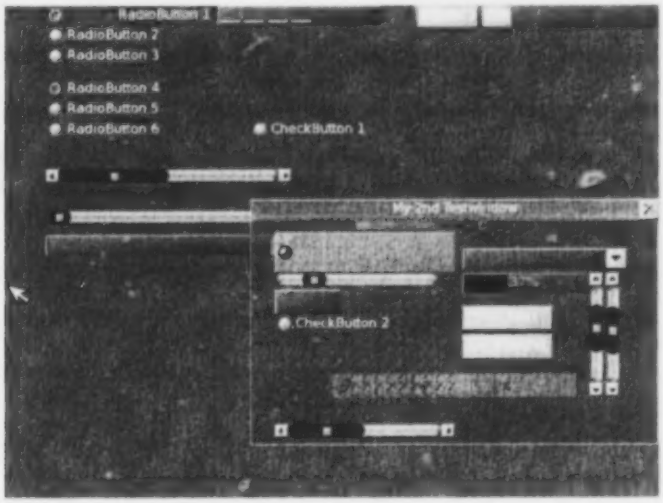

Figure 11: *Sample paraGUI interface*

The `WaveUI` class aggregates any number of lower-level graphing objects which themselves handle user interaction through controls, data representation through graphical plots and data representation through digital displays. Currently only the graphical plot classes have been implemented.

Each graphical plot object — the `GLGraph` class — can represent an arbitrary number of individual traces, encapsulated in the `GraphTrace` class. Each `GraphTrace` defines the mapping of a data flow's dimensions to various graphical elements, scales the data both in X and Y dimensions, and handles colour mapping.

Through these mechanisms, the user can create displays that represent multidimensional data in flexible and hopefully intuitive ways. For example, suppose one of the data flows we wish to display is heart rate, which consists of a stream of data triplets: timestamp, heart rate value, reliability index. The user may elect to graph 30 seconds of this data on the screen, using a scrolling display where the first five seconds of data use 25% of the screen area and the remaining 25 seconds is squeezed into the other 75%. This would provide a magnified display of the most recent data with a more condensed display of older data, keeping trends visible without details. The user may choose to map the heart rate value dimension as the Y value and the reliability index value to the trace's alpha. Thus, as the reliability

value falls, the trace becomes more and more transparent. Reliable data, on the other hand, will be plotted boldly and opaquely (figure 12).

Figure 12: *Sample graphical representation of heart rate.*

The various dimensions of the data may be used to control the trace's colour, alpha value, line width, point size or to create error bars. Dimensions may also be reused, so the same dimension may be used both for colour and for point size.

Although this user interface is implemented using OpenGL, we have also abstracted the main plot elements to support display device independence. Each `GraphTrace` object emits a stream of `TraceGraphElement` objects, which contain the device-independent graphical parameters. These objects may either to passed to a rendering engine, or for efficiency's sake, they are able to render themselves using OpenGL upon request.

2.2.4 Optimizing CPU and Storage Resources

The libraries required for implementing the various graphical user interfaces, including SDL, OpenGL, aedGUI, paraGUI and a few others, greatly increase the size of the executable and the porting effort for supporting GUARDIAN on any given platform. Since these libraries are only required for user interfaces, we created a number of conditional compilation rules and macros that allow them to be included only when required. For simple GUARDIAN applications that have no such needs, these libraries are not required and can be removed from the compilation by changing two compile time parameters.

For optimizing CPU time, we encountered some issues with the `ACE_Token` mutex. GUARDIAN makes extensive use of the ACE architecture [27], including its mutual exclusion devices. We were not aware until recently that the `ACE_Token` does not support multiple simultaneous readers. We several hours designing and implementing a multiple reader, single writer, recursive, timed, fair semaphore only to conclude that such a solution was too complex and too expensive.

Accordingly, we needed to reanalyze some critical areas of GUARDIAN, in particular the ubiquitous `DataPathManager`, to ensure the finest possible degree of granularity in its locking of critical sections. Once done, we experienced no problems in running the prototype at the required rates.

3 Future Directions

The current direction of this program is excellent, thus our recommendations are focused on continuing in the current direction and bringing to completion the projects started under this contract.

With `WaveUI` and its helper classes, we have just achieved basic functionality. This subsystem should be completed and integrated into GUARDIAN, as it will provide a good foundation for both embedded and remote data displays. For embedded systems with graphical interfaces, the user will have access to several simultaneous plots, while being careful to conserve CPU resources for real-time operations. For remote systems connected over a network, the user will be able to employ many different displays and graphs simultaneously thanks to the splitting of the real-time acquisition and display functionality.

The filesystem interface to the *datatrac* database should be implemented. We believe giving the users control over how their data appears, how it's filtered, and the organization of its various elements, will simplify their interaction with this ever-growing database.

Together with the new interface, a plugin module for exporting *datatrac* database information in HDF5 format should be created. Although *datatrac* currently supports several file formats, including zip archives, text files and various binary formats, HDF5 export will likely provide a better interface for supporting more extensive use of OpenDX.

Also, the ways that OpenDX can assist this research project need to be explored further. While we have gained some experience in using this program, the specific interaction required by the users with their acquired data needs further clarification. Implementing the previous recommendations regarding the database will provide a good foundation for this future interaction.

The creation of a script editor for GUARDIAN we consider more of a "nice to have" than a "need to have". Although tedious and somewhat error prone, the current manual method for creating network scripts is usable (see the example in annex B). The effort required to create an graphical editor is at this point difficult to justify.

Finally, we need to put together some basic PDP++ scripts that are applicable to this research project. From these we can generate the C++ which will be integrated into GUARDIAN. Even some simple examples should allow us to understand the requirements of the resulting code, in particular, the I/O, memory and CPU requirements.

Annex A
C++ Library Code

GLGraph.cpp

```
/**
 * $Revision: 1.2 $
 */

#ifdef WAVE_UI_ENABLED

#include <SDL/SDL.h>

#include "GLGraph.h"
#include "GraphTrace.h"
#include "GuardianLock.h"
#include "Guardian.h"
#include "GuardianLog.h"

GLGraph::GLGraph( const unsigned int inWidth,
                  const unsigned int inHeight,
                  const unsigned int inX,
                  const unsigned int inY )
  : maToken(), mnWidth(inWidth), mnHeight(inHeight),
    mnOriginX(inX), mnOriginY(inY)
{
  DBUG_BTRACE( "GLGraph::ctor",
               "( inWidth = %d, inHeight = %d, inX = %d, inY = %d )",
               inWidth, inHeight, inX, inY );

  ACE_Time_Value laTenthSecond;

  laTenthSecond.set( 0.1 );

  ACE_Time_Value laTenSeconds;

  laTenSeconds.set( 10.0 );

  DBUG_PRINT( DBUG_DEBUG, "test time = %f",
              laTenSeconds.msec() * 1.0e-3 );

  setPrimaryTimescale( 1.0 );
}

GLGraph::~GLGraph( void )
{
  DBUG_TRACE( "GLGraph::dtor" );
  GuardWrite guard( maToken );
}


void GLGraph::notifyDataListener( const unsigned int inSinkID,
                                  DataBlockPtr iaDataBlock )
```

```
{
  DBUG_BTRACE( "GLGraph::notifyDataListener",
               "( inSinkID = %d, iaDataBlock )", inSinkID );

  GuardWrite guard( maToken );

  /*
   * Map sink ID to the trace.
   */

  TRACE_MAP::iterator liTrace = maTraceMap.find( inSinkID );

  if ( liTrace == maTraceMap.end() ) {
    DBUG_WARN( "Received data on unknown sink ID (%d)", inSinkID );
    return;
  }

  /*
   * Update DataBlock by concatenating then trimming.
   */

  liTrace->second.mbDataChanged = true;

  DataBlock & lrDataBlock = liTrace->second.maDataBlock;

  lrDataBlock.addDataBlock( *(iaDataBlock.get()) );

  float lfFinalTimestamp = lrDataBlock.getFinalTimestamp();

  float lfTotalTime = lfFinalTimestamp - lrDataBlock.getInitialTimestamp();

  if ( lfTotalTime > maTimescaleVector[0].mfTimeWidth ) {
    //    float lfCutTime = lfFinalTimestamp - lfTotalTime;
    float lfCutTime = lfFinalTimestamp - maTimescaleVector[0].mfTimeWidth;
    lrDataBlock.cut( lfCutTime );
  }
}

void GLGraph::notifyDataListener( const unsigned int sinkID,
                                  DataPathEventPtr event )
{
  DBUG_BTRACE( "GLGraph::notifyDataListener",
               "( sinkID = %d, event )", sinkID );

  if ( event->getType() == DataPathEvent::EVENT_TIMER ) {
    drawUpdate();
  }
}

void GLGraph::drawUpdate( void )
{
```

```
  DBUG_TRACE( "GLGraph::drawUpdate" );

  GuardRead guard( maToken );

  /*
   * Run through all of the traces and update them.  Update Y values if
   * the data has changed, then draw using the representative data values.
   */

  TRACE_MAP::iterator liCurr = maTraceMap.begin();

  while( liCurr != maTraceMap.end() ) {
    if ( liCurr->second.mbDataChanged ) {
      DBUG_PRINT( DBUG_GRAPH, "update trace %d", liCurr->first );
      liCurr->second.mpTrace->updateValueScaling( liCurr->second.maDataBlock );
      liCurr->second.mbDataChanged = false;
      liCurr->second.mpTrace->graphDataBlock( liCurr->second.maDataBlock,
                                              NULL );
    }
    liCurr++;
  }
}

void GLGraph::setPrimaryTimescale( const float ifTimeWidth )
{
  DBUG_BTRACE( "GLGraph::setPrimaryTimescale", "( ifTimeWidth = %f )",
               ifTimeWidth );

  GuardWrite guard( maToken );

  maTimescaleVector.reserve( 1 );

  if ( maTimescaleVector.size() < 1 ) {
    struct TimescaleWindowInfo laTI;
    maTimescaleVector.push_back( laTI );
  }

  maTimescaleVector[0].mfTimeWidth = ifTimeWidth;
  maTimescaleVector[0].mfWindowWidth = GraphTrace::TOTAL_WINDOW_WIDTH;
  maTimescaleVector[0].mnPrecedingWindow = 0;

  TRACE_MAP::iterator curr = maTraceMap.begin();
  while( curr != maTraceMap.end() ) {
    curr->second.mpTrace->setPrimaryTimescale( ifTimeWidth );
  }
}

void GLGraph::addTimescaleWindow( const float ifTimeWidth,
                                  const float ifWindowWidth,
                                  const unsigned int inPrecedingWindow )
{
```

```
  DBUG_BTRACE( "GLGraph::addTimescaleWindow",
       "( ifTimeWidth = %f, ifWindowWidth = %f, inPrecedingWindow = %d )",
              ifTimeWidth, ifWindowWidth, inPrecedingWindow );

  GuardWrite guard( maToken );

  maTimescaleVector.reserve( maTimescaleVector.size() + 1 );

  struct TimescaleWindowInfo laTI;

  laTI.mfTimeWidth = ifTimeWidth;
  laTI.mfWindowWidth = ifWindowWidth;
  laTI.mnPrecedingWindow = inPrecedingWindow;

  maTimescaleVector.push_back( laTI );

  TRACE_MAP::iterator curr = maTraceMap.begin();
  while( curr != maTraceMap.end() ) {
    curr->second.mpTrace->addTimescaleWindow( ifTimeWidth,
                                              ifWindowWidth,
                                              inPrecedingWindow );
  }

}

GraphTrace * GLGraph::addNewTrace( const ACE_CString & isName,
                                   const unsigned int inID )
{
  DBUG_BTRACE( "GLGraph::addNewTrace", "( isName = \"%s\", inID = %d )",
               isName.c_str(), inID );

  GraphTrace * lpNewTrace = new GraphTrace();

  GuardRead guardRead( maToken );

  lpNewTrace->setDisplaySize( mnWidth, mnHeight,
                              mnOriginX, mnOriginY );

  lpNewTrace->setPrimaryTimescale( maTimescaleVector[0].mfTimeWidth );

  for(unsigned int lnIndex = 1; lnIndex < maTimescaleVector.size(); lnIndex++){
    lpNewTrace->addTimescaleWindow( maTimescaleVector[lnIndex].mfTimeWidth,
                                    maTimescaleVector[lnIndex].mfWindowWidth,
                                    maTimescaleVector[lnIndex].mnPrecedingWindow );
  }

  GuardWrite guard( maToken );

  maTraceMap[inID].mpTrace = lpNewTrace;
  maTraceMap[inID].msName  = isName;
```

```
  return( lpNewTrace );
}

GraphTrace * GLGraph::getTraceByFlowID( const unsigned int inFlowID )
{
  DBUG_BTRACE_LOW( "GLGraph::getTraceByFlowID", "( inFlowID = %d )", inFlowID );

  GuardRead guard( maToken );

  TRACE_MAP::const_iterator liCurr = maTraceMap.find( inFlowID );

  if ( liCurr == maTraceMap.end() ) {
    return( NULL );
  }
  return( liCurr->second.mpTrace );
}

void GLGraph::initializeGLAttributes( void )
{
  DBUG_TRACE_STATIC( "GLGraph::initializeGLAttributes" );

  SDL_GL_SetAttribute( SDL_GL_RED_SIZE, 5 );
  SDL_GL_SetAttribute( SDL_GL_GREEN_SIZE, 5 );
  SDL_GL_SetAttribute( SDL_GL_BLUE_SIZE, 5 );
  SDL_GL_SetAttribute( SDL_GL_DEPTH_SIZE, 16 );
  SDL_GL_SetAttribute( SDL_GL_DOUBLEBUFFER, 1 );
  SDL_GL_SetAttribute( SDL_GL_ALPHA_SIZE, 5 );
}

#endif // WAVE_UI_ENABLED
```

GLGraph.h

```
/**
 * @file
 *
 * @version
 * $Revision: 1.1 $
 */

#ifdef WAVE_UI_ENABLED

#ifndef GLGRAPH_H
#define GLGRAPH_H

#include <SDL/SDL.h>

#include <ace/Token.h>
#include <ace/SString.h>

#include <map>

#include "DataPathListener.h"    // for DataBlockPtr

#include "GraphTrace.h"

using namespace std;

/**
 * Minimialist Graphical display using OpenGL.
 *
 * This object requires SDL and assumes that SDL will be opened with
 * appropriate OpenGL initialization (i.e. SDL_OPENGL or SDL_OPENGLBIT,
 * typically the latter since this class is typically used with ParaGUI.
 *
 * The class provides a static method initializeGLAttributes() which can
 * be called <em>before</em> SDL initialization to set up GL appropriately.
 *
 * So the call pattern is:
 * <code>
 *       SDL_Init( SDL_VIDEO );
 *       GLGraph::initializeGLAttributes();
 *       SDL_SetVideoMode( ... , SDL_OPENGL );
 *       GLGraph myGraph();
 * </code>
 */

class GLGraph : public DataPathListener
{
 public:

  void drawUpdate( void );
```

```
virtual void notifyDataListener( const unsigned int sinkID,
                                 DataBlockPtr data );

virtual void notifyDataListener( const unsigned int sinkID,
                                 DataPathEventPtr event );

/**
 * Establish necessary GL attributes through the SDL_GL_SetAttribute()
 */

static void initializeGLAttributes( void );

/**
 * Update the given trace with a new DataBlock.
 *
 * @param[in] inTraceID the ID of the trace to be updated
 * @param[in] iaDataBlock the auto_ptr to the DataBlock
 */

void updateTrace( const unsigned int inTraceID,
                  DataBlockPtr iaDataBlock );

/**
 * Removes a display trace from the graph.
 *
 * @param[in] ipTrace pointer to the trace
 */

void removeTrace( GraphTrace * ipTrace );

void removeTrace( const unsigned int inTraceID );

/**
 * Adds a display trace to the graph.
 *
 * A "trace" is a single line or data signal on the display.  When this
 * method is called, a new trace slot is established and a prepared
 * GraphTrace object is returned.
 *
 * @param[in] isName the name of the trace
 * @param[in] inID the flow ID of the trace.  It is this value that will be
 * used when data comes in via notifyDataListener() to assign the data
 * to the correct trace
 *
 * @returns a pointer to the trace
 */

GraphTrace * addNewTrace( const ACE_CString & isName,
                          const unsigned int inID );
```

```
GraphTrace * getTraceByFlowID( const unsigned int inFlowID );

/**
 * Define the width of the total plot in terms of seconds of data.
 *
 * This information is propagated to all GraphTrace's aggregated by
 * the GLGraph.
 *
 * @param[in] ifTimeWidth width of the plot in seconds of data
 */

void setPrimaryTimescale( const float ifTimeWidth );

/**
 * Add additional timescales to all of the GraphTrace objects aggregated
 * by this GLGraph.
 *
 * @param[in] ifTimeWidth width of the plot in seconds of data
 * @param[in] ifWindowWidth width of the plot in terms of percentage of
 * total width of window (0.0 to 1.0)
 * @param[in] inPrecedingWindow index of the existing timescale window
 * which this new window will be inserted just to the left of
 * @ref GraphTrace::addTimescaleWindow
 */


void addTimescaleWindow( const float ifTimeWidth,
                         const float ifWindowWidth,
                         const unsigned int inPrecedingWindow );

/**
 * Create a graph with the window size and origin given in pixels.
 *  For the origin orientation, (0, 0) is upper-left corner.
 *
 * @param[in] inWidth width of the graphing window in pixels.
 * @param[in] inHeight height of the graphing window in pixels.
 * @param[in] inX X coordinate of the origin in pixels.
 * @param[in] inY Y coordinate of the origin in pixels.
 */

GLGraph( const unsigned int inWidth, const unsigned int inHeight,
         const unsigned int inX, const unsigned int inY );

virtual ~GLGraph();

private:
 mutable ACE_Token maToken;

 unsigned int mnWidth;
 unsigned int mnHeight;
 unsigned int mnOriginX;
```

```
    unsigned int mnOriginY;

    ACE_Time_Value maLeadBufferSize;

    /**
     * Stores information regarding the scaling of the data to the screen
     * over a fixed time (X) width.
     */

    struct TimescaleWindowInfo {
      float                 mfTimeWidth;
      float                 mfWindowWidth;
      unsigned int          mnPrecedingWindow;
    };

    typedef vector<struct TimescaleWindowInfo> TIMESCALE_VECTOR;

    TIMESCALE_VECTOR maTimescaleVector;

    class TraceMapInfo {
    public:
      TraceMapInfo( void )
        : mpTrace(NULL), msName(""), maDataBlock(), mbDataChanged(false),
        mpGraphVector(NULL)
        {
          mpGraphVector = new GraphTrace::TRACE_GRAPH_VECTOR;
        }
      GraphTrace * mpTrace;
      ACE_CString  msName;
      DataBlock    maDataBlock;
      bool         mbDataChanged;
      GraphTrace::TRACE_GRAPH_VECTOR * mpGraphVector;
    };

    typedef map<unsigned int, TraceMapInfo> TRACE_MAP;

    TRACE_MAP maTraceMap;
};

#endif // GLGRAPH_H

#endif // WAVE_UI_ENABLED
```

GraphTrace.cpp

```
/**
 * @file
 *
 * $Revision: 1.1 $
 */

#ifdef WAVE_ULENABLED

#include "ACE_DBUG.h"

#include "DataBlock.h"
#include "Guardian.h"
#include "GuardianException.h"
#include "TraceGraphElement.h"
#include "GraphTrace.h"

#include <iostream>

GraphTrace::GraphTrace( void )
   : maToken(), maTimescaleVector(),
     mfMaxValue( -HUGE_VALF ), mfMinValue( HUGE_VALF ),
     mfM( 1.0 ), mfB( 0.0 ),
     mnDisplayWidth(0), mnDisplayHeight(0),
     mnDisplayOriginX(0), mnDisplayOriginY(0)
{
  DBUG_TRACE_LOW( "GraphTrace::ctor" );

  setDimCount( 1 );
  setDimPrimary( 0 );
  setPrimaryTimescale( 1.0 );
  setPrimaryTraceType( (unsigned int)TT_LINES );
  setValueScaling( SF_LINEAR, SUT_LOCAL );
}

GraphTrace::~GraphTrace( void )
{
  DBUG_TRACE_LOW( "GraphTrace::dtor" );
  GuardWrite guard( maToken );
}

unsigned int GraphTrace::getDimCount( void )
{
  DBUG_TRACE_LOW( "GraphTrace::getDimCount" );
  GuardRead guard( maToken );
  return( maDimTraceModifier.size() );
}

void GraphTrace::setDimPrimary( const unsigned int inPrimaryDimIndex )
{
```

```
  DBUG_BTRACE( "GraphTrace::setDimPrimary", "( inPrimaryDimIndex = %d )",
               inPrimaryDimIndex );

  GuardWrite guard( maToken );

  if ( inPrimaryDimIndex >= getDimCount() ) {
    GEX( "Primary index larger than dimension count", E_BADINDEX );
  }

  mnPrimaryDimIndex = inPrimaryDimIndex;
}

unsigned int GraphTrace::getDimPrimary( void ) const
{
  DBUG_TRACE_LOW( "GraphTrace::getDimPrimary" );

  GuardRead guard( maToken );

  return( mnPrimaryDimIndex );
}

void GraphTrace::setPrimaryTraceType( const unsigned int inTraceTypeFlags )
{
  DBUG_BTRACE( "GraphTrace::setPrimaryTraceType", "( inTraceTypeFlags = %d )",
               inTraceTypeFlags );

  GuardWrite guard( maToken );

  /*
   * Verify trace type.
   */

  if ( inTraceTypeFlags >= (1 << 2) ) {
    DBUG_PRINT( DBUG_EX, "Trace flags setting invalid (%d > %d max)",
                inTraceTypeFlags, (1 << 4) - 1 );
    GEX( "Invalid trace type flags", E_BADARG );
  }

  mnTraceType = inTraceTypeFlags;
}

void GraphTrace::setDimCount( const unsigned int inDimCount )
{
  DBUG_BTRACE( "GraphTrace::setDimCount", "( inDimCount = %d )", inDimCount );

  GuardWrite guard( maToken );

  if ( inDimCount == maDimTraceModifier.size() ) {
    return;
  }
```

```
  if ( inDimCount < maDimTraceModifier.size() ) {
    if ( inDimCount <= mnPrimaryDimIndex ) {     // are we deleting primary?
      GEX( "New dim count would delete primary", E_BADARG );
    }

    unsigned int lnDeleteCount = maDimTraceModifier.size() - inDimCount;

    DBUG_WARN( "Deleting %d dimensions", lnDeleteCount );

    for( unsigned int lnIndex = 0; lnIndex < lnDeleteCount; lnIndex++ ) {
      maDimTraceModifier.pop_back();
    }

    return;
  }

  maDimTraceModifier.reserve( inDimCount );

  unsigned int lnAddCount = inDimCount - maDimTraceModifier.size();

  struct DimTraceModifierInfo laNewTrace;

  for( unsigned int lnIndex = 0; lnIndex < lnAddCount; lnIndex++ ) {
    maDimTraceModifier.push_back( laNewTrace );
    setDimTraceModifier( getDimCount()-1, (unsigned int)TM_NONE );
  }

  return;
}

void GraphTrace::setDimTraceModifier( const int inDimIndex,
                                      const unsigned int inTraceModifierFlags )
{
  DBUG_BTRACE( "GraphTrace::setDimTraceModifier",
       "( inDimIndex = %d, inTraceModifierFlags = %d )",
              inDimIndex, inTraceModifierFlags );

  GuardWrite guard( maToken );

  /*
   * Check index.
   */

  if ( inDimIndex >= 0 ) {
    if ( inDimIndex >= (int)maDimTraceModifier.size() ) {
      DBUG_PRINT( DBUG_EX, "Index (%d) greater than configured (%d)",
                  inDimIndex, maDimTraceModifier.size() );
      GEX( "Index greater than number of dimensions configured", E_BADINDEX );
    }
  }
```

```
  /*
   * Verify trace type.
   */

  if ( inTraceModifierFlags >= (1 << 4) ) {
    DBUG_PRINT( DBUG_EX, "Trace flags setting invalid (%d > %d max)",
                inTraceModifierFlags, (1 << 4) - 1 );
    GEX( "Invalid trace type flags", E_BADARG );
  }

  /*
   * Cycle through all selected indices (i.e. either the given index value
   * or all possible dimensions).
   */

  DIM_TRACE_MODIFIER_VECTOR::iterator curr = maDimTraceModifier.begin();

  if ( inDimIndex >= 0 ) {
    curr += inDimIndex;
  }

  while( curr != maDimTraceModifier.begin() ) {
    curr->mnTraceModifiers  = inTraceModifierFlags;
    if ( inDimIndex >= 0 ) {
      break;
    }
    curr++;
  }
}

void GraphTrace::addPrimaryColour( const float ifRed,
                                   const float ifGreen,
                                   const float ifBlue,
                                   const float ifAlpha,
                                   const bool  ibUseValueLimits,
                                   const float ifMinValue,
                                   const float ifMaxValue )
{
  DBUG_BTRACE( "GraphTrace::addPrimaryColour",
               "( ifRed = %g, ifGreen = %g, ifBlue = %g, ifAlpha = %g, ibUseValueLimi
               ifRed, ifGreen, ifBlue, ifAlpha,
               ibUseValueLimits, ifMinValue, ifMaxValue );

  addDimColour( this->getDimPrimary(),
                ifRed, ifGreen, ifBlue, ifAlpha,
                ibUseValueLimits, ifMinValue, ifMaxValue );
}


void GraphTrace::addDimColour( const int inDimIndex,
                               const float ifRed,
```

```
                                const float ifGreen,
                                const float ifBlue,
                                const float ifAlpha,
                                const bool  ibUseValueLimits,
                                const float ifMinValue,
                                const float ifMaxValue )
{
  DBUG_BTRACE( "GraphTrace::addDimColour",
               "( inDimIndex = %d, ifRed = %g, ifGreen = %g, ifBlue = %g, ifAlpha = "
               inDimIndex, ifRed, ifGreen, ifBlue, ifAlpha,
               ibUseValueLimits, ifMinValue, ifMaxValue );

  GuardWrite guard( maToken );

  /*
   * Check dimension index.
   */

  if ( inDimIndex >= 0 ) {
    if ( inDimIndex >= (int)maDimTraceModifier.size() ) {
      DBUG_PRINT( DBUG_EX, "Index (%d) greater than configured (%d)",
                  inDimIndex, maDimTraceModifier.size() );
      GEX( "Index greater than number of dimensions configured", E_BADINDEX );
    }
  }

  /*
   * Clamp colour values.
   */

  struct ColourWindow laColour;

  laColour.mfRed = (ifRed < 0.0) ? 0.0 : (ifRed > 1.0) ? 1.0 : ifRed;
  laColour.mfGreen = (ifGreen < 0.0) ? 0.0 : (ifGreen > 1.0) ? 1.0 : ifGreen;
  laColour.mfBlue = (ifBlue < 0.0) ? 0.0 : (ifBlue > 1.0) ? 1.0 : ifBlue;
  laColour.mfAlpha = (ifAlpha < 0.0) ? 0.0 : (ifAlpha > 1.0) ? 1.0 : ifAlpha;

  laColour.mbUseValueLimits = ibUseValueLimits;

  if ( ibUseValueLimits ) {
    if ( ifMinValue >= ifMaxValue ) {
      GEX( "Min value and max value are not compatible (min >= max)",
           E_BADARG );
    }
  }

  laColour.mfMinValue       = ifMinValue;
  laColour.mfMaxValue       = ifMaxValue;

  /*
   * Cycle through all selected indices (i.e. either the given index value
```

```
    * or all possible dimensions).
    */

  DIM_TRACE_MODIFIER_VECTOR::iterator curr = maDimTraceModifier.begin();

  if ( inDimIndex >= 0 ) {
    curr += inDimIndex;
  }

  while( curr != maDimTraceModifier.end() ) {
    curr->maColours.insert( laColour );
    curr->mnTraceModifiers = (TM_COLOUR | curr->mnTraceModifiers);
    if ( inDimIndex >= 0 ) {
      break;
    }
    curr++;
  }
}

void GraphTrace::setValueScaling( const SCALE_FUNCTION &     iaScalingFunction,
                                  const SCALE_UPDATE_TYPE & iaScalingType )
{
  DBUG_BTRACE( "GraphTrace::setValueScaling",
               "( iaScalingFunction = %d, iaScalingType = %d )",
               iaScalingFunction, iaScalingType );

  GuardWrite guard( maToken );

  maUpdateType = iaScalingType;
  maScaleFunction = iaScalingFunction;
}

void GraphTrace::setScaleCoeffs( const float ifM,
                                 const float ifB )
{
  DBUG_BTRACE( "GraphTrace::setScaleCoeffs",
               "( ifM = %f, ifB = %f )", ifM, ifB );

  GuardWrite guard( maToken );

  mfM = ifM;
  mfB = ifB;
}

void GraphTrace::setValuePeaks( const float ifMin,
                                const float ifMax )
{
  DBUG_BTRACE( "GraphTrace::setValuePeaks",
               "( ifMin = %f, ifMax = %f )", ifMin, ifMax );

  GuardWrite guard( maToken );
```

```
  mfMinValue = ifMin;
  mfMaxValue = ifMax;
}

void GraphTrace::updateValueScaling( DataBlock & irDB )
{
  DBUG_TRACE( "GraphTrace::updateValueScaling" );

  if ( maUpdateType == SUT_FIXED ) {
    return;
  }
  if ( irDB.getPointCount() == 0 ) {
    return;
  }

  unsigned int lnPrimaryDimIndex;
  float        lfMaxValue;
  float        lfMinValue;

  GuardRead guard( maToken );

  lnPrimaryDimIndex = mnPrimaryDimIndex;
  lfMaxValue = mfMaxValue;
  lfMinValue = mfMinValue;

  if ( irDB.getVectorWidth() <= lnPrimaryDimIndex ) {
    DBUG_WARN( "Given db is too narrow (%d dims) for this trace (min %d dims)",
               irDB.getVectorWidth(), lnPrimaryDimIndex+1 );
    return;
  }

  if ( maUpdateType == SUT_LOCAL ) {
    lfMaxValue = scaleValue( irDB.getPoint( 0, lnPrimaryDimIndex ) );
    lfMinValue = lfMaxValue;
  }

  unsigned int lnMax = irDB.getPointCount();

  for( unsigned int i = 0; i < lnMax; i++ ) {
    float lfVal = scaleValue( irDB.getPoint( i, lnPrimaryDimIndex ) );
    if ( lfVal > lfMaxValue ) {
      lfMaxValue = lfVal;
    }
    if ( lfVal < lfMinValue ) {
      lfMinValue = lfVal;
    }
  }

  /*
   * Check that the difference between min and max isn't so small as to
```

```
    * destablize the scaling calculation.  If it is, set it to a minimum value.
    */

  float lfDifference = fabs( mfMaxValue - mfMinValue );
  if ( lfDifference < 0.0001 ) {
    mfMaxValue += 0.0001;
  }

  GuardWrite guardWrite( maToken );
  mfMaxValue = lfMaxValue;
  mfMinValue = lfMinValue;
}

float GraphTrace::getMaxValue( void ) const
{
  DBUG_TRACE_LOW( "GraphTrace::getMaxValue" );

  GuardRead guard( maToken );

  return( mfMaxValue );
}

float GraphTrace::getMinValue( void ) const
{
  DBUG_TRACE_LOW( "GraphTrace::getMinValue" );

  GuardRead guard( maToken );

  return( mfMinValue );
}

void GraphTrace::setPrimaryTimescale( const float ifTimeWidth )
{
  DBUG_BTRACE( "GraphTrace::setPrimaryTimescale", "( ifTimeWidth = %f )",
               ifTimeWidth );

  if ( ifTimeWidth <= 0.0 ) {
    GEX( "Timewidth must be positive and non-zero", E_BADARG );
  }

  GuardWrite guard( maToken );

  maTimescaleVector.reserve( 1 );        // need at least 1 slot

  if ( maTimescaleVector.size() == 0 ) {
    DBUG_PRINT( DBUG_GRAPH, "initializing maTimescaleVector" );
    struct TimescaleWindow laDummy;
    maTimescaleVector.push_back( laDummy );
  }

  maTimescaleVector[0].mfTimeWidth = ifTimeWidth;
```

```
  maTimescaleVector[0].mfStartTime = 0.0;  // rightmost starts at 0 elapsed
  maTimescaleVector[0].mfWindowWidth = 1.0 * TOTAL_WINDOW_WIDTH;
  maTimescaleVector[0].mfDilation = TOTAL_WINDOW_WIDTH / ifTimeWidth;
  maTimescaleVector[0].mfTranslation = 0.0;
}

/**
 * @todo Someday I need to make sure that windows are sorted according to,
 * in order:
 * <ol>
 * <li>start time
 * <li>end time
 * <li>window width
 * </ol>
 *
 * If this was true, then the search that takes place in
 * GraphTrace::getTimescaleWindow() could be more efficient.
 */

unsigned int GraphTrace::addTimescaleWindow( const float ifTimeWidth,
                                             const float ifWindowWidth,
                                    const unsigned int inPrecedingWindow )
{
  DBUG_BTRACE( "GraphTrace::addTimescaleWindow",
       "( ifTimeWidth = %f, ifWindowWidth = %f, inPrecedingWindow = %d )",
              ifTimeWidth, ifWindowWidth, inPrecedingWindow );

  if ( ifTimeWidth <= 0.0 ) {
    GEX( "Timewidth must be positive and non-zero", E_BADARG );
  }

  GuardWrite guard( maToken );

  if ( inPrecedingWindow >= maTimescaleVector.size() ) {
    GEX( "Given index greater than maximum", E_BADINDEX );
  }

  maTimescaleVector.reserve( maTimescaleVector.size() + 1 );

  struct TimescaleWindow laWindow;

  laWindow.mfTimeWidth = ifTimeWidth;

  laWindow.mfStartTime
    = maTimescaleVector[inPrecedingWindow].mfStartTime
    + maTimescaleVector[inPrecedingWindow].mfTimeWidth;

  laWindow.mfWindowWidth = ifWindowWidth * TOTAL_WINDOW_WIDTH;

  laWindow.mfTranslation = maTimescaleVector[inPrecedingWindow].mfTranslation
    + maTimescaleVector[inPrecedingWindow].mfWindowWidth;
```

```
  laWindow.mfDilation = TOTAL_WINDOW_WIDTH * ifWindowWidth / ifTimeWidth;

  TIMESCALE_VECTOR::iterator curr = maTimescaleVector.begin();

  curr += (maTimescaleVector.size() - inPrecedingWindow - 1);

  maTimescaleVector.insert( curr, laWindow );

  return( inPrecedingWindow + 1 );
}

void GraphTrace::setDisplaySize( const unsigned int inWidth,
                                 const unsigned int inHeight,
                                 const unsigned int inOriginX,
                                 const unsigned int inOriginY )
{
  DBUG_BTRACE( "GraphTrace::setDisplaySize",
       "( inWidth = %d, inHeight = %d, inOriginX = %d, inOriginY = %d )",
              inWidth, inHeight, inOriginX, inOriginY );

  GuardWrite guard( maToken );

  mnDisplayWidth  = inWidth;
  mnDisplayHeight = inHeight;
  mnDisplayOriginX = inOriginX;
  mnDisplayOriginY = inOriginY;
}

unsigned int GraphTrace::getTimescaleWindowCount( void ) const
{
  DBUG_TRACE_LOW( "GraphTrace::getTimescaleWindowCount" );
  GuardRead guard( maToken );
  return( maTimescaleVector.size() );
}

void GraphTrace::getTimescaleWindow( const unsigned int inIndex,
                                     float & ofStartTime,
                                     float & ofEndTime,
                                     float & ofTimeWidth,
                                     float & ofWindowWidth,
                                     float & ofDilation,
                                     float & ofTranslation )
{
  DBUG_BTRACE_LOW( "GraphTrace::getTimescaleWindow",
   "( inIndex = %d, ofStartTime, ofEndTime, ofTimeWidth, ofWindowWidth, ofDilation, (
                   inIndex );

  if ( inIndex >= this->getTimescaleWindowCount() ) {
    GEX( "Given index larger than number of timescales", E_TOOLARGE );
  }
```

```
  struct TimescaleWindow laGraph = maTimescaleVector[inIndex];

  ofStartTime = laGraph.mfStartTime;
  ofEndTime   = laGraph.mfStartTime + laGraph.mfTimeWidth;
  ofTimeWidth = laGraph.mfTimeWidth;
  ofWindowWidth = laGraph.mfWindowWidth;
  ofDilation  = laGraph.mfDilation;
  ofTranslation = laGraph.mfTranslation - TOTAL_WINDOW_OFFSET;
}

void GraphTrace::graphDataBlock( const DataBlock & irDB,
                                 TRACE_GRAPH_VECTOR * opGraph )
{
  DBUG_BTRACE( "GraphTrace::graphDataBlock",
               "( irDB, opGraph %s )", (opGraph == NULL) ? "= NULL" : "" );

  GuardRead guard( maToken );

  /*
   * Pick up primary mapping first (timescale window 0).  Determine the
   * timestamp of the first point that we care about.
   */

  float lfPrimaryStartTime;
  float lfPrimaryEndTime;
  float lfPrimaryTimeWidth;
  float lfPrimaryWindowWidth;
  float lfPrimaryDilation;
  float lfPrimaryTranslation;

  getTimescaleWindow( 0, lfPrimaryStartTime, lfPrimaryEndTime,
                      lfPrimaryTimeWidth, lfPrimaryWindowWidth,
                      lfPrimaryDilation, lfPrimaryTranslation );

  DBUG_PRINT( DBUG_GRAPH, "Primary timescale: (start = %f, end = %f), (time = %f, wi
              lfPrimaryStartTime, lfPrimaryEndTime,
              lfPrimaryTimeWidth, lfPrimaryWindowWidth,
              lfPrimaryDilation, lfPrimaryTranslation );

  unsigned int lnTimescaleWindowCount = getTimescaleWindowCount();

  DBUG_PRINT( DBUG_GRAPH, "lnTimescaleWindowCount = %d",
              lnTimescaleWindowCount );

  float lfInitialTimestamp = irDB.getFinalTimestamp() - lfPrimaryTimeWidth;

  if ( lfInitialTimestamp < 0.0 ) {
    lfInitialTimestamp = 0.0;
  }
```

```
DBUG_PRINT( DBUG_GRAPH, "primary init timestamp = %f (= %f - %f)",
            lfInitialTimestamp,
            irDB.getFinalTimestamp(),
            lfPrimaryTimeWidth );

unsigned int lnDimPrimary = getDimPrimary();

/*
 * Index into the DataBlock to the start of the first timestamp
 * we care about.
 */

unsigned int lnDBIndex;
unsigned int lnDBIndexMax = irDB.getPointCount();

for( lnDBIndex = 0; lnDBIndex < lnDBIndexMax; lnDBIndex++ ) {
  if ( irDB.getTimestamp( lnDBIndex ) >= lfInitialTimestamp ) {
    break;
  }
}

/*
 * Ensure there is enough space in the opGraph (if given) to store the
 * data.  Space required is the size of the DataBlock (# pts) minus
 * the already-skipped points (the value of lnDBIndex).
 */

unsigned int lnSizeReqd = irDB.getPointCount() - lnDBIndex;

if ( opGraph != NULL ) {
  unsigned int lnCapacity = opGraph->capacity();
  if ( lnCapacity < lnSizeReqd ) {
    DBUG_PRINT( DBUG_GRAPH, "expanding opGraph size by %d elements",
                lnSizeReqd - lnCapacity );
    opGraph->reserve( lnSizeReqd );
    TraceGraphElement dummy;
    for( unsigned int i = lnCapacity; i < lnSizeReqd; i++ ) {
      opGraph->push_back( dummy );
    }
  }
}

/*
 * Run through all of the data points.  We start at the current value
 * of lnDBIndex, since this has been forwarded to the start of our
 * overall window.
 */

unsigned int lnGraphIndex = 0;

for( ; lnDBIndex < lnDBIndexMax; lnDBIndex++ ) {
```

```
/*
 * Set the initial timescale values.
 */

float lfStartTime = lfPrimaryStartTime;
float lfEndTime   = lfPrimaryEndTime;
float lfTranslation = lfPrimaryTranslation;
float lfDilation    = lfPrimaryDilation;
float lfTimeWidth   = lfPrimaryTimeWidth;

unsigned int lnTimeWindowIndex = 0;

/*
 * Get the current data timestamp.  Convert to time window centric
 * time (i.e. subtract the initial timestamp of the window).
 */

float lfCurrentTime = irDB.getTimestamp( lnDBIndex );

lfCurrentTime -= lfInitialTimestamp;

DBUG_ASSERT( lfCurrentTime > -0.00001 );     // just a check

/*
 * Run through timescale windows, looking for the smallest non-primary
 * window which bounds the current point.
 */

for( unsigned int i = 1; i < lnTimescaleWindowCount; i++ ) {
  float lfInnerStartTime;
  float lfInnerEndTime;
  float lfInnerTimeWidth;
  float lfDummy;
  float lfInnerTranslation;
  float lfInnerDilation;

  getTimescaleWindow( i, lfInnerStartTime, lfInnerEndTime,
                      lfInnerEndTime, lfDummy,
                      lfInnerDilation, lfInnerTranslation );

  if ( (lfCurrentTime >= lfInnerStartTime) &&
       (lfCurrentTime <= lfInnerEndTime) ) {
    if ( lfInnerTimeWidth <= lfTimeWidth ) {
      DBUG_ASSERT( false );
      lnTimeWindowIndex = i;
      lfStartTime = lfInnerStartTime;
      lfEndTime   = lfInnerEndTime;
      lfTimeWidth = lfInnerTimeWidth;
      lfTranslation = lfInnerTranslation;
      lfDilation    = lfInnerDilation;
```

```
    }
  }
}

/*
 * Now, use the found timescale window to scale this data point.
 */

TraceGraphElement laGraph;

/*
 * First, scale the timestamp and store.
 */

DBUG_PRINT( DBUG_GRAPH, "tsw[%d] = (%f, %f) --> (%f, %f)",
            lnTimeWindowIndex, lfStartTime, lfEndTime,
            lfDilation, lfTranslation );

laGraph.mfX = (lfCurrentTime - lfStartTime) * lfDilation + lfTranslation;

/*
 * Next, scale the value of the primary dimension.
 */

float lfCurrentValue = irDB.getPoint( lnDBIndex, lnDimPrimary );
float lfYtemp = scaleValue( lfCurrentValue );
laGraph.mfY = mapValue( lfYtemp );

/*
 * Run through all of the dimensions, performing plotting as required.
 */

unsigned int lnDimCount = irDB.getVectorWidth();
unsigned int lnDimMax   = getDimCount();

if ( lnDimCount > lnDimMax ) {
  lnDimCount = lnDimMax;
}

for( unsigned int lnDimIndex = 0; lnDimIndex < lnDimCount; lnDimIndex++ ) {
  unsigned int lnModifiers
    = maDimTraceModifier[lnDimIndex].mnTraceModifiers;

  DBUG_PRINT( DBUG_GRAPH, "mod[%d] = %d", lnDimIndex, lnModifiers );

  if ( lnModifiers != TM_NONE ) {
    if ( lnModifiers & TM_DOTSIZE ) {
    }
    if ( lnModifiers & TM_COLOUR ) {
      mapColours( lfCurrentValue, lnDimIndex,
                  laGraph.mfRed, laGraph.mfGreen, laGraph.mfBlue,
```

```
                        laGraph.mfAlpha );
            }
            if ( lnModifiers & TM_ERROR_BARS ) {
            }
            if ( lnModifiers & TM_THICKNESS ) {
            }
          }
        }

        DBUG_PRINT( DBUG_GRAPH,
                    "laGraph: X = %f, Y = %f, colour = (%f, %f, %f, %f)",
                    laGraph.mfX, laGraph.mfY,
                    laGraph.mfRed, laGraph.mfGreen, laGraph.mfBlue,
                    laGraph.mfAlpha );

        if ( opGraph != NULL ) {
          (*opGraph)[lnGraphIndex] = laGraph;
          DBUG_PRINT( DBUG_GRAPH, "assigned laGraph to %d", lnGraphIndex );
          lnGraphIndex++;
        }
      }
    }

    void GraphTrace::mapColours( const float ifValue,
                                 const unsigned int inDimIndex,
                                 float & ofRed,
                                 float & ofGreen,
                                 float & ofBlue,
                                 float & ofAlpha )
    {
      DBUG_BTRACE_LOW( "GraphTrace::mapColours",
       "( ifValue = %f, inDimIndex = %d, ofRed, ofGreen, ofBlue, ofAlpha )",
                        ifValue, inDimIndex );

      GuardRead guard( maToken );

      if ( inDimIndex >= maDimTraceModifier.size() ) {
        GEX( "Index too large", E_TOOLARGE );
      }

      const COLOUR_SET & lrColours = maDimTraceModifier[inDimIndex].maColours;

      COLOUR_SET::const_iterator liCurr = lrColours.begin();

      ofRed   = 0.0;
      ofGreen = 0.0;
      ofBlue  = 0.0;
      ofAlpha = 0.0;

      /*
       * Cycle through all of the ColourWindow's in the COLOUR_SET.  If we find
```

```
     * one which uses the limits, if our value is within its limits, set
     * the colour and stop the search.  If we find a value which uses the
     * limits but our value is outside of those, keep searching.  If we
     * find a value which has no limits, set the colour but keep searching
     * in case there is a more restrictive limit-enforcing colour which will
     * take precedence.
     */

    while( liCurr != lrColours.end() ) {
      if ( liCurr->mbUseValueLimits ) {
        if ( ifValue >= liCurr->mfMinValue ) {
          if ( ifValue <= liCurr->mfMaxValue ) {
            ofRed   = liCurr->mfRed;
            ofGreen = liCurr->mfGreen;
            ofBlue  = liCurr->mfBlue;
            ofAlpha = liCurr->mfAlpha;
            break;
          }
        }
      } else {
        ofRed   = liCurr->mfRed;
        ofGreen = liCurr->mfGreen;
        ofBlue  = liCurr->mfBlue;
        ofAlpha = liCurr->mfAlpha;
      }

      liCurr++;
    }
}

float GraphTrace::mapValue( const float ifValue )
{
  DBUG_BTRACE_LOW( "GraphTrace::mapValue", "( ifValue = %f )", ifValue );

  float lfValue = 0.0;

  lfValue = (ifValue - mfMinValue) / (mfMaxValue - mfMinValue)
    * TOTAL_WINDOW_WIDTH - TOTAL_WINDOW_OFFSET;

  return( lfValue );
}

float GraphTrace::scaleValue( const float ifValue )
{
  DBUG_BTRACE_LOW( "GraphTrace::scaleValue", "( ifValue = %f )", ifValue );

  float lfValue = 0.0;

  switch( maScaleFunction ) {
  case GraphTrace::SF_LINEAR:
    lfValue = (ifValue - mfB) / mfM;
```

```
      break;
    case GraphTrace::SF_LOG:
      ifValue = log( ifValue - mfB ) / mfM;
      break;
    default:
      DBUG_ASSERT( false );
    }

  return( ifValue );
}

#endif // WAVE_UI_ENABLED
```

GraphTrace.h

```
/**
 * @file
 *
 * $Revision: 1.1 $
 */

#ifdef WAVE_UI_ENABLED

#ifndef TRACEINFO_H
#define TRACEINFO_H

#include <set>
#include <vector>

#include <ace/Token.h>

#include "TraceGraphElement.h"

class DataBlock;

/**
 * Represents the information used in rendering a trace, including colours,
 * sizes, dimensions both screen and data. The object MT-safe since it
 * could potentially be accessed by multiple threads simultaneously.
 */

class GraphTrace
{
 public:

  /**
   * Different ways any particular dimension scalar can be applied to the
   * graph. All of these flags can be OR'd to create a composite display.
   */

  enum TRACE_TYPE_FLAGS {
    TT_NONE   = ( 0 << 0 ),      //!< Dimension not used
    TT_POINTS = ( 1 << 0 ),      //!< Individual points (time, value)
    TT_LINES  = ( 1 << 1 ),
                //!< Dimensionless points with connecting 0th order lines
  };

  /**
   * Different ways any particular dimension scalar can be applied to the
   * primary dimension graph. All of these flags can be OR'd to create a
   * composite display.
   */

  enum TRACE_MODIFIER_FLAGS {
```

```
  TM_NONE      = ( 0 << 0 ),     //!< Dimension not used.
  TM_DOTSIZE = ( 1 << 0 ),     //!< Changes the dot size on primary plot
  TM_COLOUR   = ( 1 << 1 ),     //!< Changes colour of primary plot
  TM_ERROR_BARS = ( 1 << 2 ), //!< Adds symmetric error bars to primary
  TM_THICKNESS   = ( 1 << 3 )  //!< Controls thickness of dot/line of primary
};

/**
 * Different ways scales can be computed or updated.
 */

enum SCALE_UPDATE_TYPE {
  SUT_FIXED = 0,        //!< Use fixed mapping
  SUT_LOCAL = 1,        //!< Use local max/min of displayed data
  SUT_PEAKS = 2,         //!< Use non-local max/min as discoverd
};

/**
 * Scale function for mapping value to screen value (Y axis) coordinates.
 */

enum SCALE_FUNCTION {
  SF_LINEAR = 0,        //!< Use linear mapping, Y = (y - b) / m
  SF_LOG = 1            //!< Use log mapping, Y = log (y - b) / m
};

/**
 * Set the primary dimension.
 */

void setDimPrimary( const unsigned int inPrimaryDimIndex );

unsigned int getDimPrimary( void ) const;

/**
 * Set the primary dimension plot type.
 *
 * @inTraceTypeFlags type of display, an OR'd list of
 */

void setPrimaryTraceType( const unsigned int inTraceTypeFlags );

/**
 * Set secondary dimension trace modifier types.
 *
 * @param inDimIndex index between 0 and #dims-1 (if negative, setting
 * applies to all dimensions except the primary)
 * @param inTraceModifierFlags type of display, an OR'd list of
 * TRACE_MODIFIER_FLAGS
 */
```

```
void setDimTraceModifier( const int inDimIndex,
                          const unsigned int inTraceModifierFlags );

/**
 * Add a colour setting to the primary dimension.
 *
 * @param ifRed the desired amount of red, between 0.0 and 1.0
 * @param ifGreen the desired amount of green, between 0.0 and 1.0
 * @param ifBlue the desired amount of blue, between 0.0 and 1.0
 * @param ifAlpha the desired amount of transparency, between 0.0
 * (fully transparent) and 1.0 (fully opaque).
 * @param ibUseValueLimits    if true, then use the given value limits,
 * otherwise always apply this colour
 * @param ifMinValue the minimum data value for which this colour should
 * be used
 * @param ifMaxValue the maximum data value for which this colour should
 * be used
 */

void addPrimaryColour( const float ifRed,
                       const float ifGreen,
                       const float ifBlue,
                       const float ifAlpha,
                       const bool  ibUseValueLimits = false,
                       const float ifMinValue = 0.0,
                       const float ifMaxValue = 0.0 );

/**
 * Add a colour and optional value range for the given dimension if
 * being used as a colour map index.
 *
 * @param inDimIndex index of the target dimension (if negative, applies
 * to all dimensions)
 * @param ifRed the desired amount of red, between 0.0 and 1.0
 * @param ifGreen the desired amount of green, between 0.0 and 1.0
 * @param ifBlue the desired amount of blue, between 0.0 and 1.0
 * @param ifAlpha the desired amount of transparency, between 0.0
 * (fully transparent) and 1.0 (fully opaque).
 * @param ibUseValueLimits    if true, then use the given value limits,
 * otherwise always apply this colour
 * @param ifMinValue the minimum data value for which this colour should
 * be used
 * @param ifMaxValue the maximum data value for which this colour should
 * be used
 */

void addDimColour( const int   inDimIndex,
                   const float ifRed,
                   const float ifGreen,
                   const float ifBlue,
                   const float ifAlpha,
```

```
                    const bool  ibUseValueLimits = false,
                    const float ifMinValue = 0.0,
                    const float ifMaxValue = 0.0 );

/**
 * Set the value (Y axis) scaling protocol.
 *
 * @param iaScalingFunction the method whereby value limits are to be
 * recomputed
 * @param iaScalingType the type of scaling to be used
 */

void setValueScaling( const SCALE_FUNCTION     & iaScalingFunction,
                      const SCALE_UPDATE_TYPE & iaScalingType );

/**
 * Set the value (Y axis) scaling coefficients.
 *
 * @param ifM initial multiplier (slope if linear) value
 * @param ifB initial offset (Y intercept if linear) value
 */

void setScaleCoeffs( const float ifM,
                     const float ifB );

/**
 * Set the peak (max/min) values for the trace.
 *
 * @param ifMin initial minimum Y value
 * @param ifMax initial maximum Y value
 */

void setValuePeaks( const float ifMin, const float ifMax );

/**
 * Scale the given value using the current slope, offset and mapping
 * function.
 */

float scaleValue( const float ifValue );

/**
 * Map the given scaled value (scaled via scaleValue()) to a normalized
 * plotting (Y coordinate) position using the peaks settings and values,
 * slope, offset, and mapping function.
 */

float mapValue( const float ifValue );

/**
 * Set the physical dimensions of the plotting display in pixels.
```

```
 *
 * @param inWidth width of the display/window
 * @param inHeight height of the display/window
 * @param inOriginX X position (horizontal) to be used for drawing this
 *  trace
 * @param inOriginY Y position (vertical) to be used for drawing this trace
 */

void setDisplaySize( const unsigned int inWidth,
                     const unsigned int inHeight,
                     const unsigned int inOriginX,
                     const unsigned int inOriginY );

/**
 * Set the primary timescale which is applied to the entire window.
 *
 * @param ifTimeWidth number of seconds of data to be displayed over the
 * entire window.
 */

void setPrimaryTimescale( const float ifTimeWidth );

/**
 * Add additional timescale windows.
 *
 * Timescale windows consist of two values:
 * <ol>
 * <li>the width of the window relative to data time, in seconds</li>
 * <li>the width of the window relative to the total graphical (pixel)
 * window, as a percentage between 0.0 and 1.0.
 * </ol>
 *
 * The placement of timescale windows needs additional explanation.  The
 * primary timescale window defines the basic mapping over the entire
 * graphical window.  Additional timescale windows are inserted to the left
 * (data scrolls from right to left) of the existing timescale window whose
 * index is given as inPrecedingWindow.  If this is the first timescale
 * window to be defined, it will be placed to the most extreme right of
 * the overall window.
 *
 * For example, if you want the basic, overall window to be 25 seconds
 * wide, with incoming data displayed at high resolution for 1/3 of the
 * screen, then accumulated at a much reduced resolution over the remaining,
 * you would use the following commands:
 * \code
 *    GraphTrace laT1;
 *
 *    laT1.setPrimaryTimescale( 25.0 );
 *
 *    unsigned int laIndex = laT1.addTimescaleWindow( 5.0, 0.3, 0 );
 *
```

```
     *     laTI.addTimescaleWindow( 20.0, 0.7, laIndex );
     * \endcode
     *
     * @param ifTimeWidth number of seconds of data to be displayed over this
     * new timescale window
     * @param ifWindowWidth percentage (between 0.0 and 1.0) of the total window
     * which is to be dedicated to this timescale window
     * @param inPrecedingWindow index number of a previously defined window
     * to which this one is to be joined to (set to 0 if this is the first
     * timescale window to be added)
     *
     * @returns the index of the new timescale window
     */

    unsigned int addTimescaleWindow( const float ifTimeWidth,
                                     const float ifWindowWidth,
                                     const unsigned int inPrecedingWindow = 0 );

    /**
     * Set the number of dimensions in the trace.
     */

    void setDimCount( const unsigned int inDimCount );

    /**
     * Returns the number of known dimensions.
     */

    unsigned int getDimCount( void );

    typedef std::vector<TraceGraphElement> TRACE_GRAPH_VECTOR;

    /**
     * Given a complete screen DataBlock, this method plots the data.  Either
     * it generates the high-level plot information and returns it via
     * the opGraph vector, or if the vector is NULL, makes the OpenGL calls
     * directly.  (OpenGL calls are not made if the vector is not NULL.)
     *
     * @param irDB input DataBlock representing the entire screen
     * @param opGraph output high-level plot vector, which if not NULL will
     * be filled with the necessary information for creating the graph.
     */

    void graphDataBlock( const DataBlock & irDB, TRACE_GRAPH_VECTOR * opGraph );

    float getMaxValue( void ) const;

    float getMinValue( void ) const;

    /**
     * Update the value scales (min and max Y values) from the given DataBlock
```

```
     * according to the set update algorithm.
     */

    void updateValueScaling( DataBlock & irDB );

    /**
     * Return the number of timescale windows.
     */

    unsigned int getTimescaleWindowCount( void ) const;

    /**
     * Retrieve the limits and transformation values for a particular
     * timescale window.
     *
     * @param ofTimeWidth number of seconds of data represented by this
     * particular window
     * @param ofWindowWidth physical width of the window represented in
     * normalized window coordinates (i.e. the entire window is (-1, +1),
     * with this window being some subset of that value, being
     * (ofWindowWidth, ofWindowWidth + ofTranslation)
     *
     * To map a timestamp X via the timescale, confirm that X is within
     * the window ( ofStartTime <= X <= ofEndTime), then the plotting X
     * coordinate value = (X - ofStartTime) * ofDilation + ofTranslation.
     */

    void getTimescaleWindow( const unsigned int inIndex,
                             float & ofStartTime,
                             float & ofEndTime,
                             float & ofTimeWidth,
                             float & ofWindowWidth,
                             float & ofDilation,
                             float & ofTranslation );

    /**
     * Return the colours which are assigned to a given value of the
     * given dimension.
     *
     * @param ifValue the value of the dimension
     * @param inDimIndex the index of the dimension
     * @param ifRed returned red coordinate
     * @param ifGreen returned green coordinate
     * @param ifBlue returned blue coordinate
     * @param ifAlpha returned alpha coordinate
     */

    void mapColours( const float ifValue,
                     const unsigned int inDimIndex,
                     float & ifRed,
                     float & ifGreen,
```

```
                    float & ifBlue,
                    float & ifAlpha );

  /**
   * The coordinate system we plot into is the orthonormal one used by
   * OpenGL, where the X axis and Y axis run from -1.0 to +1.0, giving
   * a total width of 2.0.
   */

  static const float TOTAL_WINDOW_WIDTH = 2.0;

  /**
   * The coordinate system we plot into is the orthonormal one used by
   * OpenGL, where the X axis and Y axis run from -1.0 to +1.0, giving
   * a total width of 2.0, which is shifted negatively by an offset of 1.0.
   */

  static const float TOTAL_WINDOW_OFFSET = 1.0;

  /**
   * A sanity-checking maximum for the number of timescale windows.
   * Primarily used for allocating caches.
   */

  static const unsigned int MAX_TIMESCALE_WINDOW_COUNT = 100;

  GraphTrace( void );

  virtual ~GraphTrace( void );

private:
  mutable ACE_Token maToken;

  /**
   * Information which makes up the individual timescale windows.
   *
   * Timescale windows have their time values defined in time elapsed from
   * the right of the screen (so, time elapsed during the display of a
   * given point) which is the reverse of time of acquisition.
   *
   * So, time elapsed = 0.0 is the very rightmost point of the screen (which
   * is the newest displayed data point). Time elapsed = 5.0 indicates a
   * data point which has been displayed for 5 seconds.
   *
   * If the actual timestamp of the newest plotted data point is T, then
   * the timestamps of data displayed in a TimescaleWindow of mfStartTime
   * = ST and a mfTimeWidth of W would be those timestamps S such that
   * T - ST - W <= S <= T - ST.
   */

  struct TimescaleWindow {
```

```
  float        mfTimeWidth;              //!< Number of seconds in the window
  float        mfStartTime;              //!< When the window starts
  float        mfWindowWidth;            //!< Width of the screen assigned
  float        mfTranslation;
  float        mfDilation;
};

typedef std::vector<struct TimescaleWindow> TIMESCALE_VECTOR;

TIMESCALE_VECTOR maTimescaleVector;

struct ColourWindow {
  bool         mbUseValueLimits;
  float        mfMinValue;
  float        mfMaxValue;
  float        mfRed;
  float        mfGreen;
  float        mfBlue;
  float        mfAlpha;
};

struct ltColourWindow {
  bool operator()( const struct ColourWindow & a,
                   const struct ColourWindow & b )
  {
    if ( !a.mbUseValueLimits && !b.mbUseValueLimits ) {
      return( true );
    } else if ( a.mbUseValueLimits && !b.mbUseValueLimits ) {
      return( false );
    } else if ( !a.mbUseValueLimits && b.mbUseValueLimits ) {
      return( true );
    }
    return( a.mfMinValue < b.mfMinValue );
  }
};

typedef std::set<struct ColourWindow, ltColourWindow> COLOUR_SET;

class DimTraceModifierInfo {
public:
  unsigned int    mnTraceModifiers;  //!< from TRACE_MODIFIER_FLAGS
  COLOUR_SET      maColours;         //!< Applicable colour structs
  DimTraceModifierInfo( void ) : mnTraceModifiers(TM_NONE), maColours() { }
};

typedef std::vector<DimTraceModifierInfo> DIM_TRACE_MODIFIER_VECTOR;

DIM_TRACE_MODIFIER_VECTOR maDimTraceModifier;

unsigned int          mnTraceType;     //!< Flags from TRACE_TYPE_FLAGS
```

```
                                        //! Index of the main trace dimension
  unsigned int          mnPrimaryDimIndex;

  SCALE_UPDATE_TYPE     maUpdateType;     //!< How to update the value scaling
  float                 mfMaxValue;       //!< Max Y axis value
  float                 mfMinValue;       //!< Min Y axis value
  SCALE_FUNCTION        maScaleFunction;  //!< Mapping from value to Y axis
  float                 mfM;              //!< Value scaling multiplier
  float                 mfB;              //!< Value scaling offset

  /**
   * Width in pixels of the display area used by this trace.
   */

  unsigned int          mnDisplayWidth;

  /**
   * Height in pixels of the display area used by this trace.
   */

  unsigned int          mnDisplayHeight;

  /**
   * X origin (horizontal) of the display window.
   */

  unsigned int          mnDisplayOriginX;

  /**
   * Y origin (vertical) of the display window.
   */

  unsigned int          mnDisplayOriginY;
};

#endif // TRACEINFO_H

#endif // WAVE_UI_ENABLED
```

TraceGraphElement.cpp

```
/**
 * @file
 *
 * $Revision: 1.1 $
 */

#ifdef GL_ENABLED
#include <GL/glu.h>
#endif //GL_ENABLED

#include "TraceGraphElement.h"

#include "ACE_DBUG.h"
#include "Guardian.h"

TraceGraphElement::TraceGraphElement( void )
  : mfX(0.0), mfY(0.0), mfRadius(0.0), mfErrBar(0.0),
    mfRed(0.0), mfGreen(0.0), mfBlue(0.0), mfAlpha(0.0)
{
  DBUG_TRACE_LOW( "TraceGraphElement::ctor" );
}

TraceGraphElement::~TraceGraphElement( void )
{
  DBUG_TRACE_LOW( "TraceGraphElement::dtor" );
}

void TraceGraphElement::renderGL( void )
{
  DBUG_TRACE_LOW( "TraceGraphElement::renderGL" );

#ifdef GL_ENABLED
  glColor4f( mfRed, mfGreen, mfBlue, mfAlpha );
  glVertex2f( mfX, mfY );
#endif
}
```

TraceGraphElement.h

```
/**
 * @file
 *
 * $Revision: 1.1 $
 */

#ifndef TRACEGRAPHELEMENT_H
#define TRACEGRAPHELEMENT_H

/**
 * Represents a high-level description of a single element within a data
 * plot or graph.  Can also render itself using GL if enabled.
 */

class TraceGraphElement
{
 public:
  TraceGraphElement( void );
  virtual ~TraceGraphElement( void );
  void renderGL( void );
  float mfX;
  float mfY;
  float mfRadius;        //!< circle/dot radius
  float mfErrBar;        //!< size of error bar(s)
  float mfRed;
  float mfGreen;
  float mfBlue;
  float mfAlpha;
};

#endif // TRACEINFO_H
```

WaveUI.cpp

```
/**
 * $Revision: 1.2 $
 */

#ifdef WAVE_UI_ENABLED

#include <ace/SString.h>
#include <ace/Env_Value_T.h>

#include <pgwindow.h>

#include "WaveUI.h"
#include "WaveUIMO.h"
#include "ACE_DBUG.h"
#include "DataBlock.h"
#include "DataPathFlow.h"
#include "DataPathManager.h"
#include "DataPathTransform.h"
#include "GraphTrace.h"
#include "Guardian.h"
#include "GuardianException.h"
#include "PropertyList.h"
#include "Timer.h"
#include "TimerServer.h"

WaveUI * WaveUI::mpInstance = (WaveUI *)NULL;

void WaveUI::getParameters( PropertyList & rmPropList )
{
  DBUG_TRACE_LOW_STATIC( "WaveUI::getParameters" );

  DataPathTransform::getParameters( rmPropList );

  rmPropList.setStringProperty( Guardian::param, "name", "WaveUI" );
  rmPropList.setStringProperty( Guardian::param, "device", "user interface" );
  rmPropList.setStringProperty( Guardian::param, "width", "800" );
  rmPropList.setStringProperty( Guardian::param, "height", "600" );
}

#ifdef WORKING_WAVE
void WaveUI::buttonCallback( WaveButton * mpButton )
{
  DBUG_TRACE_LOW( "WaveUI::buttonCallback" );

  WUIbuttonCallback_MO * mo = new WUIbuttonCallback_MO( this, mpButton );

  queueMethodObject( mo );
}
#endif
```

```
void WaveUI::postConfirmDialog_( const char * ipTitle )
{
  DBUG_BTRACE( "WaveUI::postConfirmDialog_",
               "( ipTitle = \"%s\" )", ipTitle );

  /*
   * Only one confirmation can be active at a time.
   */
#ifdef WORKING_WAVE

  if ( mpConfirmFrame != NULL ) {
    return;
  }

  /*
   * Create frame.
   */

  DBUG_PRINT( DBUG_DEBUG, "_ValueFrameWidth = %d", _ValueFrameWidth );
  DBUG_PRINT( DBUG_DEBUG, "_ValueFrameHeight = %d", _ValueFrameHeight );
  DBUG_PRINT( DBUG_DEBUG, "_ConfirmFrameWidth = %d", _ConfirmFrameWidth );
  DBUG_PRINT( DBUG_DEBUG, "_ConfirmFrameHeight = %d", _ConfirmFrameHeight );
  DBUG_PRINT( DBUG_DEBUG, "_ConfirmFrameOriginX = %d", _ConfirmFrameOriginX );
  DBUG_PRINT( DBUG_DEBUG, "_ConfirmFrameOriginY = %d", _ConfirmFrameOriginY );

  mpConfirmFrame = new aedFrame();
  mpConfirmFrame->create( mpValueFrame,
                          aedRect( _ConfirmFrameOriginX,
                                   _ConfirmFrameOriginY,
                                   _ConfirmFrameWidth,
                                   _ConfirmFrameHeight ),
                          ipTitle );

  /*
   * Create buttons.
   */

  DBUG_ASSERT_DETAIL( mpOkButton == NULL, false, "" );
  DBUG_ASSERT_DETAIL( mpCancelButton == NULL, false, "" );

  mpOkButton = new aedButton();

  mpOkButton->create( mpConfirmFrame,
                      aedRect( 0, 0,
                               _ConfirmButtonWidth, _ConfirmButtonHeight ),
                      "OK" );

  mpOkButton->connect( MOUSE_LBUTTON_UP,
                       aedCallBack( WaveUI::buttonCallbackStatic ) );
```

```
  mpCancelButton = new aedButton();

  mpCancelButton->create( mpConfirmFrame,
                          aedRect( _ConfirmButtonWidth*2, 0,
                                   _ConfirmButtonWidth, _ConfirmButtonHeight ),
                          "Cancel" );

  mpCancelButton->connect( MOUSE_LBUTTON_UP,
                           aedCallBack( WaveUI::buttonCallbackStatic ) );

#endif
}

void WaveUI::buttonCallback_( void * mpButton )
{
  DBUG_TRACE( "WaveUI::buttonCallback_" );
}

void WaveUI::saveCommand( const char * isCommand )
{
  DBUG_BTRACE( "WaveUI::saveCommand", "( isCommand = \"%s\" )",
               isCommand );

  if ( msControlFileName.length() > 0 ) {
    FILE * lpFile = fopen( msControlFileName.c_str(), "w" );
    if ( lpFile == NULL ) {
      GEX( "Could not open control file", E_IO );
    }
    fprintf( lpFile, "%s\n", isCommand );
    fclose( lpFile );
  }
}

#ifdef WORKING_WAVE

void WaveUI::graphData_( const unsigned int mSinkID,
                         const WaveUI::GRAPH_TYPE mType )
{
  DBUG_BTRACE( "WaveUI::graphData_",
               "( mSinkID = %d, mType = %d )", mSinkID, (int)mType );

  _UpdateValues = false;

  mpCanvas = new Canvas();

  char lpTmp[81];

  snprintf( lpTmp, 81, "%s (%s)",
            _ValueRow[mSinkID].pLabel->getCaption().c_str(),
            _ValueRow[mSinkID].msUnits.c_str() );
```

```
  mpCanvas->create( mpDesktop,
                    aedRect( _GraphFrameOriginX, _GraphFrameOriginY,
                             _GraphFrameWidth, _GraphFrameHeight ),
                    lpTmp );

  mpCanvas->connect( MOUSE_LBUTTON_UP,
                     aedCallBack(WaveUI::buttonCallbackStatic) );

  mpCanvas->bringToTop();
  mpCanvas->renderAll();

  mpGraph = new Graph( mpCanvas );

  maGraphType = mType;
  mnGraphSinkID = mSinkID;

  mpGraph->updateGraphStructure();

  mpDesktop->renderAll();
}
#endif

#ifdef WORKING_WAVE
void WaveUI::buttonCallbackStatic( void * mpButton, void * ) {
  DBUG_TRACE_STATIC( "WaveUI::buttonCallbackStatic" );
  WaveUI * lpUI = WaveUI::instance();
  if ( lpUI != NULL ) {
    lpUI->buttonCallback( (WaveButton *)mpButton );
  }
}
#endif

WaveUI *  WaveUI::instance()
{
  return( mpInstance );
}

WaveUI::WaveUI( const ACE_CString & isControlFile )
  : DataPathTransform(), maApplication(), mpGraph(NULL),
    mbGraphicsInitialized(false), WUITask_(new WaveUITask(this)),
    mbSDLinitialized(false), mpCleanupHandler(new WaveUICleanup(this))
{
  DBUG_BTRACE( "WaveUI::ctor", "( isControlFile = \"%s\" )",
               isControlFile.c_str() );

  GuardWrite guard( _token );

  if ( mpInstance != NULL ) {
    GEX( "Attempt to instantiate a second WaveUI", E_INTERNAL );
  }
```

```
  setUpdateRate( 10.0 );

  mpInstance = this;
}

WaveUI::WaveUI( PropertyList & pl )
  : DataPathTransform(pl), maApplication(), mpGraph(NULL),
    mbGraphicsInitialized(false), WUITask_(new WaveUITask(this)),
    mbSDLinitialized(false), mpCleanupHandler(new WaveUICleanup(this))
{
  DBUG_TRACE( "WaveUI::ctor(pl)" );

  GuardWrite guard( _token );
  if ( mpInstance != NULL ) {
    GEX( "Attempt to instantiate a second WaveUI", E_INTERNAL );
  }

  ACE_CString lsString;

  pl.getStringProperty( Guardian::param, "width", lsString );

  mnDisplayWidth = Guardian::strtol( lsString );

  DBUG_PRINT( DBUG_DEBUG, "mnDisplayWidth = %d (%s)", mnDisplayWidth,
              lsString.c_str() );

  pl.getStringProperty( Guardian::param, "height", lsString );

  mnDisplayHeight = Guardian::strtol( lsString );

  pl.getStringProperty( Guardian::param, "update", lsString );

  mfUpdateRate = Guardian::strtof( lsString );

  mpInstance = this;
}

WaveUI::~WaveUI()
{
  DBUG_TRACE( "WaveUI::dtor" );
}

void WaveUI::mInitializeGraphics( void )
{
  DBUG_TRACE( "WaveUI::mInitializeGraphics" );

  if ( mbGraphicsInitialized ) {
    return;
  }
  mbGraphicsInitialized = true;
```

```
  setUpdateRate( 10.0 );

  maApplication.LoadTheme( "default" );

  GLGraph::initializeGLAttributes();

  DBUG_PRINT( DBUG_DEBUG, "width = %d, height = %d",
              mnDisplayWidth, mnDisplayHeight );

  if ( !maApplication.InitScreen( mnDisplayWidth, mnDisplayHeight, 16,
                                  SDL_DOUBLEBUF | SDL_HWSURFACE |
                                  SDL_ANYFORMAT | SDL_OPENGLBLIT ) ) {
    GEX( "Could not initialize SDL/paraGUI screen.", E_IO );
  }

  mbSDLinitialized = true;

  /*
   * Create the graph.  Then create sufficient graph traces for all currently
   * known flows to be plotted (if any).
   */

  mpGraph = new GLGraph( mnDisplayWidth, mnDisplayHeight, 0, 0 );
}

void WaveUI::mInitializeTraces( void )
{
  DBUG_TRACE( "WaveUI::mInitializeTraces" );

  /*
   * Cycle through our invoking PropertyList, creating a trace
   * and a sink for each given trace value.
   */

  PropertyList & lrProp = this->getDetailsWriteable();

  unsigned int lnTraceCount = lrProp.getPropertyCount( "trace" );

  DBUG_PRINT( DBUG_GRAPH, "creating %d traces", lnTraceCount );

  for( unsigned int lnIndex = 0; lnIndex < lnTraceCount; lnIndex++ ) {
    char lpIndexString[81];

    /*
     * Retrieve the trace name.
     */

    snprintf( lpIndexString, 81, "%d", lnIndex );

    ACE_CString lsTraceName;
```

```
    if ( lrProp.getStringProperty( "trace", lpIndexString, lsTraceName ) != 1){
      DBUG_ASSERT( false );
    }

    /*
     * Make the flow.
     */

    char lpTmp[81];
    snprintf( lpTmp, 81, "%s %d", lsTraceName.c_str(), lnIndex );

    DataPathFlow * lpNewFlow = new DataPathFlow( lpTmp, "",
                                                 DataPathFlow::SINK,
                                          DataPathFlow::PERIODICITY_UNSPECIFIED,
                                                 DataPathFlow::PHYSICAL,
                                                 1.0,
                                                 false );
    lpNewFlow->setDPTransID( getID() );
    lpNewFlow->setVectorWidth( 1 );
    lpNewFlow->initialize();

    /*
     * Make the trace.
     */

    DBUG_ASSERT( mpGraph != NULL );

    GraphTrace * lpTrace = mpGraph->addNewTrace( lsTraceName,
                                                 lpNewFlow->getID() );

    /*
     * Finish configuring the trace (colours, dimension mappings, etc.).
     */

    lpTrace->setDimCount( 1 );
    lpTrace->setDimPrimary( 0 );

  }
}

void WaveUI::setUpdateRate( const float ifRate )
{
  DBUG_BTRACE( "WaveUI::setUpdateRate", "( ifRate = %f )", ifRate );

  float lfRate = ifRate;

  if ( lfRate <= 0.0 ) {
    lfRate = 0.0001;
  }

  maUpdateInterval.set( 1.0 / lfRate );
```

```
}

int WaveUI::open( void * )
{
  DBUG_TRACE( "WaveUI::open" );

  // This is a good time to register any I/O handles, file descriptors,
  // sockets, etc. with the Reactor.  Notice that the handler given is the
  // WUITask_, since this is our adaptor which includes the ACE_Event_Handler
  // interface.
  //
  //  ACE_Reactor::instance()->register_handler( _TriggerPipe[0], WUITask_,
  //                                             ACE_Event_Handler::READ_MASK );

  // We *must* register ourselves to receive events from the Reactor
  // to deal with initialization from the Thread manager.

  ACE_Reactor::instance()->register_handler( WUITask_,
                                             ACE_Event_Handler::READ_MASK );

  // Now, call our adaptor's "activate" method, which will start up our
  // new thread with the given parameters.  Return the return code as our
  // final step.

  return( WUITask_->activate( Guardian::ThreadSchedule ) );
}

int WaveUI::svc( void )
{
  DBUG_TRACE( "WaveUI::svc" );

  // The following is the first bit of code to run in our new WaveUI
  // thread, so we immediately name the thread.

  DBUG_THREAD( "WUI" );
  ACE_Thread_Manager::instance()->wait_on_exit( 1 );

  // Register our ACE_Cleanup with the ACE_Object_Manager so that if the
  // system is shut down we'll be given a chance to check Guardian and
  // determine the termination state.

  ACE_Object_Manager::instance()->at_exit( mpCleanupHandler );

  //
  // Activate the graphics system.
  //

  try {
    mInitializeGraphics();

    //
```

```
    // Initialize the update timer.
    //

    ACE_Future<Timer *> timer_future =
      TIMER_SERVER::instance()->getTimer( Timer::T_PERIODIC, maUpdateInterval,
                                          this );

    while( !GUARDIAN::instance()->isTerminate() ) {
      ACE_Auto_Basic_Ptr<ACE_Method_Request> mo( activation_queue_.dequeue() );

      if ( (*mo).call() == -1 ) {
        break;
      }
    }
  }
  catch( GuardianException & ge ) {
    DBUG_PRINT( DBUG_EX, "Caught exception .. exiting" );
  }

  if ( mbSDLinitialized ) {
    SDL_Quit();
  }

  DBUG_PRINT( DBUG_TRACE, "exiting" );

  return( 0 );
}

void WaveUI::notifyDataListener( const unsigned int sinkID,
                                 DataPathEventPtr event )
{
  DBUG_BTRACE( "WaveUI::notifyDataListener",
               "( sinkID = %d, event )", sinkID );

  WUInotifyDataListener_MO * mo = new WUInotifyDataListener_MO( this, sinkID,
                                                                event );

  queueMethodObject( mo );
}

void WaveUI::notifyDataListener_( const unsigned int sinkID,
                                  DataPathEventPtr event )
{
  DBUG_TRACE( "WaveUI::notifyDataListener_(event)" );

  /*
   * If it's a TIMER event, do a graphics update.
   */

  if ( event->getType() == DataPathEvent::EVENT_TIMER ) {
```

```
    GuardRead guard( *token_ );

    /*
     * But don't call SDL commands if SDL is not first initialized.
     */

    if ( mbSDLinitialized ) {

      /*
       * ...and don't call SDL commands if we've been told to terminate,
       * since the window(s) may already be closed.
       */

      if ( !GUARDIAN::instance()->isTerminate() ) {

        /*
         * Notify the graph that we need to do an update (that is, send
         * it the timer event).
         */

        mpGraph->notifyDataListener( sinkID, event );

        /*
         * Flip the buffers.
         */

        PG_Window::BulkBlit();
        SDL_GL_SwapBuffers();
        SDL_Flip( maApplication.GetScreen() );

        /*
         * Run through any events.
         */

        SDL_Event laEvent;
        while ( SDL_PollEvent( &laEvent ) ) {
          maApplication.PumpIntoEventQueue( &laEvent ); //for paragui processng
          if ( laEvent.type == SDL_QUIT ) {
            GUARDIAN::instance()->shutdown();
            break;
          }
        }
      }
    }
  }

  // Then invoke the parent class handler.

  DataPathTransform::notifyDataListener_( sinkID, event );
}
```

```
int WaveUI::handle_input( ACE_HANDLE handle )
{
  DBUG_TRACE( "WaveUI::handle_input" );

  WUIhandle_input_MO * mo = new WUIhandle_input_MO( this, handle );

  queueMethodObject( mo );

  return( 0 );
}

int WaveUI::handle_input_( ACE_HANDLE handle )
{
  DBUG_TRACE( "WaveUI::handle_input_" );

  // Perform whatever input processing is required for this handle.
  return( 0 );
}

int WaveUI::handle_close( ACE_HANDLE handle )
{
  DBUG_TRACE( "WaveUI::handle_close" );

  WUIhandle_close_MO * mo = new WUIhandle_close_MO( this, handle );

  queueMethodObject( mo );

  return( 0 );
}

int WaveUI::handle_close_( ACE_HANDLE handle )
{
  DBUG_TRACE( "WaveUI::handle_close_" );

  // Perform whatever close processing is required for this handle.
  return( 0 );
}

void WaveUI::dump( void )
{
  this->dump( 0 );
}

void WaveUI::dump( unsigned int level )
{
  GuardRead guard( *token_ );
}

void WaveUI::notifyDataListener( const unsigned int sinkID,
                                 DataBlockPtr data )
{
```

```
  DBUG_BTRACE( "WaveUI::notifyDataListener",
               "( sinkID = %d, data )", sinkID );

  WUInotifyDataListener_MO * mo = new WUInotifyDataListener_MO( this, sinkID,
                                                                data );

  queueMethodObject( mo );
}

int
WaveUI::queueMethodObject( ACE_Method_Request * mo )
{
  DBUG_BTRACE_LOW( "WaveUI::queueMethodObject",
                   "( mo = 0x%08x )", (unsigned int)mo );

  ACE_Time_Value timeout( GLock::instance()->getTimeoutTime() );
  ACE_Time_Value * timeoutPtr = &timeout;

  if ( timeout == ACE_Time_Value::zero ) {
    timeoutPtr = 0;
  }

  return( activation_queue_.enqueue( mo, timeoutPtr ) );
}

void WaveUI::initializeFlows( void )
{
  DBUG_TRACE( "WaveUI::initializeFlows" );
  WUIinitializeFlows_MO * mo = new WUIinitializeFlows_MO( this );

  queueMethodObject( mo );
}

void WaveUI::initializeFlows_( void )
{
  DBUG_TRACE( "WaveUI::initializeFlows_" );

  mInitializeTraces();
}

void WaveUI::notifyDataListener_( const unsigned int sinkID,
                                  DataBlockPtr data )
{
  DBUG_TRACE( "WaveUI::notifyDataListener_(data)" );

  if ( data.get() == NULL ) {
    GEX( "Received NULL DataBlock", E_BADARG );
  }

  if ( mpGraph != NULL ) {
    mpGraph->notifyDataListener( sinkID, data );
```

```
  }

#ifdef WORKING_WAVE
  if ( mpGraph != NULL ) {
    if ( sinkID == mnGraphSinkID ) {
      if ( maGraphType == WaveUI::GRAPH_CURRENT ) {
        mpGraph->addData( data );
      }
    }
  }
#endif
}

#endif // WAVE_UI_ENABLED
```

WaveUI.h

```
/**
 * Graphical Waveform User Interface .
 *
 * $Revision: 1.1 $
 *
 */

#ifdef WAVE_UI_ENABLED

#ifndef WAVEUI_H
#define WAVEUI_H

#include <ace/Activation_Queue.h>          // required for active object
#include <ace/Synch.h>                     // required for multithreading
#include <ace/Task.h>                      // required for multithreading

#include <paragui.h>
#include <pgapplication.h>

#include "ACE_DBUG.h"
#include "DataPathTransform.h"
#include "GLGraph.h"
#include "GuardianLog.h"

#include <map>

using namespace std;

class WaveUITask;
class WaveUICleanup;

class WaveUI : public DataPathTransform
{
 public:
  /**
   * Not exactly a Singleton.  There may only be once WaveUI in a system,
   * and it is accessible through this method, however, if this method is
   * called before an instance is created, it will simply return NULL rather
   * than create an WaveUI.
   */

  static WaveUI * instance( void );

  /**
   * This method is the paragui value button callback function.
   */

  //  static void buttonCallbackStatic( void * button, void * data );
```

```
/**
 * This method is the WaveUI::buttonCallbackStatic callback function to
 * cross from class method to specific object method.
 */

//  void buttonCallback( AquatsButton * button );

static void getParameters( PropertyList & rmPropList );

virtual void dump( void );
virtual void dump( unsigned int level );

/**
 * DataPathTransform (proxy) methods.
 */

virtual void initializeFlows();

virtual void notifyDataListener( const unsigned int sinkID,
                                 DataBlockPtr data );

virtual void notifyDataListener( const unsigned int sinkID,
                                 DataPathEventPtr event );

/**
 * Task methods.
 */

int open( void * );

int svc( void );

int queueMethodObject( ACE_Method_Request * mo );

/**
 * Event handler (proxy) methods.
 */

int handle_input( ACE_HANDLE handle );
int handle_close( ACE_HANDLE handle );

/**
 * Constructor.
 *
 * @param isControlFile the name (full path) of the file into which
 * the user interface writes the control words.  If the user selects
 * "save", then "save" is written into the file.  If "discard", then
 * "discard" is written into it.
 */

WaveUI( const ACE_CString & isControlFile );
```

```
    WaveUI( PropertyList & param );
    virtual ~WaveUI();

    void saveCommand( const char * );

    void setUpdateRate( const float ifHertz );

  private:

    void mInitializeGraphics( void );

    void mInitializeTraces( void );

    /** Width of the physical display in pixels.                    */
    unsigned int mnDisplayWidth;

    /** Height of the physical display in pixels.                   */
    unsigned int mnDisplayHeight;

    /** Colour depth of the physical display in bits/pixel.         */
    unsigned int mnDisplayDepth;

    PG_Application maApplication;

    GLGraph *       mpGraph;

    bool            mbGraphicsInitialized;

    float           mfUpdateRate;

    ACE_Time_Value maUpdateInterval;

    ACE_CString msControlFileName;

    void postConfirmDialog_( const char * );
    void unpostConfirmDialog_( void );

    void initializeValues_( void );
    void initializeFixed_( void );

    void updateCurrentTime_( void );

    /**
     * DataPathTransform (servant) methods.
     */

    friend class WUIinitializeFlows_MO;
    friend class WUInotifyDataListener_MO;

    virtual void initializeFlows_();
```

```
    virtual void notifyDataListener_( const unsigned int sinkID,
                                      DataBlockPtr data );

    virtual void notifyDataListener_( const unsigned int sinkID,
                                      DataPathEventPtr event );

    /**
     * Create a graphical display of the given button according to the given
     * type.
     */

    //  void graphData_( const unsigned int sinkID, const GRAPH_TYPE type );

    friend class WUIbuttonCallback_MO;
    void buttonCallback_( void * button );

    /**
     * Event handler (servant) methods.
     */

    friend class WUIhandle_input_MO;
    friend class WUIhandle_close_MO;

    int handle_input_( ACE_HANDLE handle );
    int handle_close_( ACE_HANDLE handle );

    /**
     * Queue required for active object.
     */

    ACE_Activation_Queue  activation_queue_;

    /**
     * ACE_Task adaptor.
     */

    WaveUITask * WUITask_;

    static WaveUI * mpInstance;

    ACE_Token _token;

    bool mbSDLinitialized;

    WaveUICleanup * mpCleanupHandler;
};

/**
 * Adapt the ACE_Task interface to the WaveUI.
 *
 * This class allows WaveUI to run in its own thread as an
```

```
 * active object.  It does require manual locking of any resources which
 * may be shared between threads.
 *
 * ACE_Task also implements event handler.  If you don't want the event
 * handler functions, just delete the handle_* methods.
 */

class WaveUITask : public ACE_Task<ACE_MT_SYNCH>
{
 public:
  WaveUITask( WaveUI * agg ) : agg_(agg)
    { }
  int open( void * v ) {
    assert( agg_ != NULL );
    return( agg_->open( v ) );
  }
  int svc( void ) {
    assert( agg_ != NULL );
    return( agg_->svc() );
  }
  virtual int handle_input( ACE_HANDLE handle ) {
    return( agg_->handle_input( handle ) );
  }
  virtual int handle_close( ACE_HANDLE handle, ACE_Reactor_Mask ) {
    return( agg_->handle_close( handle ) );
  }
  virtual ~WaveUITask()
    { }
 private:
  WaveUI * agg_;
};

/**
 * Adapt the ACE_Cleanup interface to the WaveUI to allow cleanup by the
 * ACE_Object_Manager.
 */

class WaveUICleanup : public ACE_Cleanup
{
 public:

  /**
   * Send an EOD event to the WaveUI to allow it to detect and process
   * termination.
   */

  virtual void cleanup( void * param = 0 ) {
    //    SDL_Quit();
    cerr << "cleanup called" << endl;
    assert( mpAgg != NULL );
    DataPathEventPtr lCompleteEvent( new DataPathEvent(
```

```
                                    DataPathEvent::EVENT_END_OF_DATA ) );
    mpAgg->notifyDataListener( 0, lCompleteEvent );
  }
  WaveUICleanup( WaveUI * ipAgg ) : mpAgg( ipAgg ) {
    assert( mpAgg != NULL );
  }
  virtual ~WaveUICleanup() {
  }
 private:
  WaveUI * mpAgg;
};


#endif // WAVEUI_H

#endif // WAVE_UI_ENABLED
```

WaveUIMO.h

```
/**
 * Method objects for the WaveUI.
 *
 * $Revision: 1.1 $
 */

#ifdef WAVE_UI_ENABLED

#ifndef WAVEUIMO_H
#define WAVEUIMO_H

#include <ace/Method_Request.h>

#include "WaveUI.h"

class WUIinitializeFlows_MO : public ACE_Method_Request {
 public:
  WUIinitializeFlows_MO( WaveUI * waveUI )
    {
      DBUG_TRACE_LOW( "WUIinitializeFlows_MO::ctor" );
      waveUI_ = waveUI;
    }

  ~WUIinitializeFlows_MO()
    {
      DBUG_TRACE_LOW( "WUIinitializeFlows_MO::dtor" );
    }

  virtual int call( void )
    {
      DBUG_TRACE_LOW( "WUIinitializeFlows_MO::call" );
      this->waveUI_->initializeFlows_();
      return( 0 );
    }
 private:
  WaveUI * waveUI_;
};

class WUIhandle_input_MO : public ACE_Method_Request {
 public:
  WUIhandle_input_MO( WaveUI * waveUI, ACE_HANDLE handle )
    {
      DBUG_TRACE_LOW( "WUIhandle_input_MO::ctor" );
      waveUI_ = waveUI;
      handle_ = handle;
    }

  ~WUIhandle_input_MO()
    {
```

```
    DBUG_TRACE_LOW( "WUIhandle_input_MO::dtor" );
  }

 virtual int call( void )
  {
    DBUG_TRACE_LOW( "WUIhandle_input_MO::call" );
    this->waveUI_->handle_input_( handle_ );
    return( 0 );
  }
private:
 WaveUI *       waveUI_;
 ACE_HANDLE             handle_;
};

#ifdef WORKING_WAVE
class WUIbuttonCallback_MO : public ACE_Method_Request {
public:
 WUIbuttonCallback_MO( WaveUI * waveUI, WaveButton * button )
  {
    DBUG_TRACE_LOW( "WUIbuttonCallback_MO::ctor" );
    waveUI_ = waveUI;
    button_ = button;
  }

 ~WUIbuttonCallback_MO()
  {
    DBUG_TRACE_LOW( "WUIbuttonCallback_MO::dtor" );
  }

 virtual int call( void )
  {
    DBUG_TRACE_LOW( "WUIbuttonCallback_MO::call" );
    this->waveUI_->buttonCallback_( button_ );
    return( 0 );
  }
private:
 WaveUI *       waveUI_;
 WaveButton * button_;
};
#endif

class WUIhandle_close_MO : public ACE_Method_Request {
public:
 WUIhandle_close_MO( WaveUI * waveUI, ACE_HANDLE handle )
  {
    DBUG_TRACE_LOW( "WUIhandle_close_MO::ctor" );
    waveUI_ = waveUI;
    handle_ = handle;
  }

 ~WUIhandle_close_MO()
```

```
    {
      DBUG_TRACE_LOW( "WUIhandle_close_MO::dtor" );
    }

  virtual int call( void )
    {
      DBUG_TRACE_LOW( "WUIhandle_close_MO::call" );
      this->waveUI_->handle_close_( handle_ );
      return( 0 );
    }
 private:
  WaveUI *        waveUI_;
  ACE_HANDLE      handle_;
};

class WUInotifyDataListener_MO : public ACE_Method_Request {
 public:
  WUInotifyDataListener_MO( WaveUI *  st,
                            unsigned int          sinkID,
                            DataBlockPtr          data )
    {
      DBUG_BTRACE_LOW( "WUInotifyDataListener_MO::ctor",
                       "( sinkID = %d, data )", sinkID );
      st_ = st;
      sinkID_ = sinkID;
      data_   = data;
      isData_ = true;
    }
  WUInotifyDataListener_MO( WaveUI *  st,
                            unsigned int          sinkID,
                            DataPathEventPtr      event )
    {
      DBUG_BTRACE_LOW( "WUInotifyDataListener_MO::ctor",
                       "( sinkID = %d, event )", sinkID );
      st_ = st;
      sinkID_ = sinkID;
      event_  = event;
      isData_ = false;
    }

  ~WUInotifyDataListener_MO()
    {
      DBUG_TRACE_LOW( "WUInotifyDataListener_MO::dtor" );
    }

  virtual int call( void )
    {
      DBUG_TRACE_LOW( "WUInotifyDataListener_MO::call" );
      if ( isData_ ) {
        st_->notifyDataListener_( sinkID_, data_ );
      } else {
```

```
        st_->notifyDataListener_( sinkID_, event_ );
      }
      return( 0 );
    }
  private:
   WaveUI *        st_;
   unsigned int            sinkID_;
   bool                    isData_;
   DataBlockPtr            data_;
   DataPathEventPtr        event_;
};

#endif // WAVEUIMO_H

#endif // WAVE_UI_ENABLED
```

Annex B
Configuration Script

waveui.txt

```
#
# WaveUI simulation and test.
#

#
# Create the physical (so, within Guardian, virtual) environment we are
# measuring.
#

air             [ dpe=DataPathVirtual,
                  name="submarine air",
                  device="environment",
                  action="create"
                ];

air::CO2        [ name="carbon dioxide",
                  direction="source",
                  vectorWidth=1,
                  periodicity="timeseries",
                  rate=200.0,
                  action="create"
                ];

air::O2         [ name="oxygen",
                  direction="source",
                  vectorWidth=1,
                  periodicity="timeseries",
                  rate=200.0,
                  action="create"
                ];

air::VOC        [ name="volatile organic compounds",
                  direction="source",
                  vectorWidth=1,
                  periodicity="timeseries",
                  rate=1.0,
                  action="create"
                ];

air::CO         [ name="carbon monoxide",
                  direction="source",
                  vectorWidth=1,
                  periodicity="timeseries",
                  rate=1.0,
                  action="create"
                ];

air::H2         [ name="hydrogen",
                  direction="source",
```

```
                    vectorWidth=1,
                    periodicity="timeseries",
                    rate=200.0,
                    action="create"
                  ];

air::temp         [ name="temperature",
                    direction="source",
                    vectorWidth=1,
                    periodicity="timeseries",
                    rate=200.0,
                    action="create"
                  ];

air::press        [ name="pressure",
                    direction="source",
                    vectorWidth=1,
                    periodicity="timeseries",
                    rate=200.0,
                    action="create"
                  ];

air::humidity     [ name="humidity",
                    direction="source",
                    vectorWidth=1,
                    periodicity="timeseries",
                    rate=200.0,
                    action="create"
                  ];

#
# We also monitor the battery state of the system
#

battery           [ dpe=DataPathVirtual,
                    name="AQUATS battery",
                    device="battery",
                    Manufacturer.name="Tri-M Systems",
                    Manufacturer.serialno="199392-332",
                    Manufacturer.productno="Tri-M Batt 32V NiMH",
                    action="create"
                  ];

battery::volts    [ name="battery voltage",
                    direction="source",
                    vectorWidth=1,
                    periodicity="timeseries",
                    rate=200.0,
                    action="create"
                  ];
```

```
#
# We also monitor the temperature of the CPU.
#

CPU             [ dpe=DataPathVirtual,
                  name="Pentium III, 500 MHz",
                  device="CPU",
                  action="create",
                  Manufacturer.name="Lippert Cool Road Runner III PC/104+",
                  Manufacturer.productno="CRIII",
                  Manufacturer.serialno="99-291-293"
                ];

CPU::temp       [ name="temperature",
                  direction="source",
                  vectorWidth=1,
                  periodicity="timeseries",
                  rate=1.0,
                  action="create"
                ];

#
# Create the sensors we are using to monitor the environment.
#

# OXYGEN SENSOR

O2sensor        [ dpe=DataPathVirtual,
                  name="FireSystems O2 Sensor",
                  device="sensor",
                  action="create"
                ];

O2sensor::in    [ name="Vent In",
                  direction="sink",
                  vectorWidth=1,
                  periodicity="timeseries",
                  rate=200.0,
                  action="create"
                ];

O2sensor::out   [ name="O2 Out",
                  direction="source",
                  vectorWidth=1,
                  periodicity="timeseries",
                  rate=200.0,
                  action="create"
                ];

# CARBON DIOXIDE SENSOR
```

```
CO2sensor          [ dpe=DataPathVirtual,
                     name="FireSystems CO2 Sensor",
                     device="sensor",
                     action="create"
                   ];

CO2sensor::in      [ name="Vent In",
                     direction="sink",
                     vectorWidth=1,
                     periodicity="timeseries",
                     rate=200.0,
                     action="create"
                   ];

CO2sensor::out     [ name="CO2 Out",
                     direction="source",
                     vectorWidth=1,
                     periodicity="timeseries",
                     rate=200.0,
                     action="create"
                   ];

CO2sensor::control [ name="CO2 Control Input",
                     direction="sink",
                     vectorWidth=1,
                     periodicity="periodic",
                     rate=0.01,
                     action="create"
                   ];

CO2sensor::watchdog [ name="CO2 Watchdog",
                      direction="source",
                      vectorWidth=1,
                      periodicity="timeseries",
                      rate=1.0,
                      action="create"
                    ];

# VOC SENSOR

VOCsensor          [ dpe="DataPathVirtual",
                     name="Malamute Inc. VOC Sensor",
                     device="sensor",
                     action="create"
                   ];

VOCsensor::in      [ name="Vent In",
                     direction="sink",
                     vectorWidth=1,
                     periodicity="timeseries",
                     rate=1.0,
```

```
                    action="create"
                  ];

VOCsensor::out    [ name="VOC Out",
                    direction="source",
                    vectorWidth=1,
                    periodicity="timeseries",
                    rate=1.0,
                    action="create"
                  ];

# CO SENSOR

COsensor          [ dpe=DataPathVirtual,
                    name="First Alert CO Sensor",
                    device="sensor",
                    action="create"
                  ];

COsensor::in      [ name="Vent In",
                    direction="sink",
                    vectorWidth=1,
                    periodicity="timeseries",
                    rate=1.0,
                    action="create"
                  ];

COsensor::out     [ name="CO Out",
                    direction="source",
                    vectorWidth=1,
                    periodicity="timeseries",
                    rate=1.0,
                    action="create"
                  ];

# H2 SENSOR

H2sensor          [ dpe=DataPathVirtual,
                    name="Microcore H2 Sensor",
                    device="sensor",
                    action="create"
                  ];

H2sensor::in      [ name="Vent In",
                    direction="sink",
                    vectorWidth=1,
                    periodicity="timeseries",
                    rate=1.0,
                    action="create"
                  ];
```

```
H2sensor::out   [ name="H2 Out",
                  direction="source",
                  vectorWidth=1,
                  periodicity="timeseries",
                  rate=1.0,
                  action="create"
                ];

# TEMPERATURE/HUMIDITY SENSOR

TempSensor      [ dpe=DataPathVirtual,
                  name="Microcore Temperature/Humidity Probe",
                  device="sensor",
                  action="create"
                ];

TempSensor::tempIn      [ name="Vent In",
                          direction="sink",
                          vectorWidth=1,
                          periodicity="timeseries",
                          rate=1.0,
                          action="create"
                        ];

TempSensor::humidIn     [ name="Vent In",
                          direction="sink",
                          vectorWidth=1,
                          periodicity="timeseries",
                          rate=1.0,
                          action="create"
                        ];

TempSensor::tempOut     [ name="Temperature Out",
                          direction="source",
                          vectorWidth=1,
                          periodicity="timeseries",
                          rate=1.0,
                          action="create"
                        ];

TempSensor::humidOut    [ name="Humidity Out",
                          direction="source",
                          vectorWidth=1,
                          periodicity="timeseries",
                          rate=1.0,
                          action="create"
                        ];

# PRESSURE SENSOR

PressSensor     [ dpe=DataPathVirtual,
```

```
                    name="Validyne Absolute Pressure Transducer",
                    device="sensor",
                    action="create"
                  ];

PressSensor::in   [ name="Vent In",
                    direction="sink",
                    vectorWidth=1,
                    periodicity="timeseries",
                    rate=1.0,
                    action="create"
                  ];

PressSensor::out  [ name="Pressure Out",
                    direction="source",
                    vectorWidth=1,
                    periodicity="timeseries",
                    rate=1.0,
                    action="create"
                  ];

#
# Connect our virtual sensors to our virtual sensands.
#

air::CO2 -> CO2sensor::in        [ name="CO2 connection",
                                   type="virtual",
                                   label="CO2_sensor",
                                   action="create"
                                 ];

air::O2 -> O2sensor::in          [ name="O2 connection",
                                   type="virtual",
                                   label="O2_sensor",
                                   action="create"
                                 ];

air::VOC -> VOCsensor::in        [ name="VOC connection",
                                   type="virtual",
                                   label="VOC_sensor",
                                   action="create"
                                 ];

air::CO -> COsensor::in          [ name="CO connection",
                                   type="virtual",
                                   label="CO_sensor",
                                   action="create"
                                 ];

air::H2 -> H2sensor::in          [ name="H2 connection",
                                   type="virtual",
```

```
                                    label="H2_sensor",
                                    action="create"
                                  ];

air::temp -> TempSensor::tempIn   [ name="Temperature connection",
                                    type="virtual",
                                    label="Temp_sensor",
                                    action="create"
                                  ];

air::humidity -> TempSensor::humidIn    [ name="Humidity connection",
                                          type="virtual",
                                          label="Humid_sensor",
                                          action="create"
                                        ];

air::press -> PressSensor::in     [ name="Pressure connection",
                                    type="virtual",
                                    label="Pressure_sensor",
                                    action="create"
                                  ];

#
# Create the ADC/DAC interface simulation device.
#

adc               [ dpe=AnalogDigitalSim,
                    name="analog to digital",
                    device="adc simulator",

                    analogInChannel.0="O2:sine,10,30,5",
                    analogInChannel.1="CO2:sine,20,5,5",
                    analogInChannel.2="VOC:sine,2,10,5",
                    analogInChannel.3="CO:sine,15,20,10",
                    analogInChannel.4="H2:sine,100,1,1",
                    analogInChannel.5="temperature:sine,10,32,1",
                    analogInChannel.6="pressure:sine,20,22,5",
                    analogInChannel.7="humidity:sine,1,100,1",
                    analogInChannel.8="battery voltage:sine,10,16,16",
                    analogInChannel.9="CPU temperature:sine,5,70,10",
                    analogIn.rate=100,
                    analogIn.count=32, # 32 ADC channels

                    digitalOut.count=4,
                    digitalOut.rate=1,
                    digitalOutChannel.0="power down",
                    digitalOutChannel.1="reset O2",

                    digitalIn.count=8,
                    digitalIn.rate=1,
```

```
    digitalInChannel.5="O2 sensor status"

    action="create"
  ];

#
# Get access to some of the virtual analog inputs.
#

adc::AIchan00   [ name="AI channel 00", rate=200.0, action="assign" ];

adc::AIchan01   [ name="AI channel 01", rate=200.0, action="assign" ];

adc::AIchan02   [ name="AI channel 02", rate=200.0, action="assign" ];

adc::AIchan03   [ name="AI channel 03", rate=200.0, action="assign" ];

adc::AIchan04   [ name="AI channel 04", rate=200.0, action="assign" ];

adc::AIchan05   [ name="AI channel 05", rate=200.0, action="assign" ];

adc::AIchan06   [ name="AI channel 06", rate=200.0, action="assign" ];

adc::AIchan07   [ name="AI channel 07", rate=200.0, action="assign" ];

adc::AIchan08   [ name="AI channel 08", rate=200.0, action="assign" ];

adc::AIchan12   [ name="AI channel 12", rate=200.0, action="assign" ];

#
# Get access to some of the outputs.
#

adc::O2out      [ name="O2", rate=200.0, action="assign" ];

adc::CO2out     [ name="CO2", rate=200.0, action="assign" ];

adc::COout      [ name="CO", rate=200.0, action="assign" ];

adc::H2out      [ name="H2", rate=200.0, action="assign" ];

adc::VOCout     [ name="VOC", rate=200.0, action="assign" ];

adc::PressOut   [ name="pressure", rate=200.0, action="assign" ];

adc::TempOut    [ name="temperature", rate=200.0, action="assign" ];

adc::HumidOut   [ name="humidity", rate=200.0, action="assign" ];

adc::BatteryOut [ name="battery voltage", rate=200.0, action="assign" ];
```

```
adc::CPUTempOut [ name="CPU temperature", rate=200.0, action="assign" ];

#
# Get access to some of the virtual digital inputs and outputs.
#

adc::DIchan05    [ name="DI channel 05",  rate=1.0,  action="assign" ];

adc::DOchan00    [ name="DO channel 00",  rate=1.0,  action="assign" ];

adc::O2status    [ name="O2 sensor status", rate=1.0, action="assign" ];

adc::O2reset     [ name="reset O2", rate=1.0, action="assign" ];

adc::powerDown   [ name="power down", rate=1.0, action="assign" ];

#
# Connect sensor outputs to analog inputs.
#

O2sensor::out            -> adc::AIchan00
  [ name="connection",  type="virtual", label="conn",   action="create"  ];

CO2sensor::out           -> adc::AIchan01
  [ name="connection",  type="virtual", label="conn",   action="create"  ];

VOCsensor::out           -> adc::AIchan02
  [ name="connection",  type="virtual", label="conn",   action="create" ];

COsensor::out            -> adc::AIchan03
  [ name="connection",  type="virtual", label="conn",   action="create"  ];

CPU::temp                -> adc::AIchan04
  [ name="connection",  type="virtual", label="conn",   action="create"  ];

H2sensor::out            -> adc::AIchan05
  [ name="connection",  type="virtual", label="conn",   action="create"  ];

TempSensor::tempOut      -> adc::AIchan06
  [ name="connection",  type="virtual", label="conn",   action="create"  ];

TempSensor::humidOut     -> adc::AIchan07
  [ name="connection",  type="virtual", label="conn",   action="create"  ];

PressSensor::out         -> adc::AIchan08
  [ name="connection",  type="virtual", label="conn",   action="create"  ];

battery::volts           -> adc::AIchan12
  [ name="connection",  type="virtual", label="conn",   action="create"  ];

#
```

```
# Connect sensors to digital inputs and outputs.
#

CO2sensor::watchdog       -> adc::DIchan05
  [ name="connection",  type="virtual", label="conn",    action="create"  ];

adc::DOchan00             -> CO2sensor::control
  [ name="connection",  type="virtual", label="conn",    action="create"  ];

#
# Create calibration transforms and access flows.
#

O2Calibrate       [ dpe=DataPathArithmetic,
                    name="O2 Calibration",
                    device="computation",

                    initializer="",
                    arithmetic="output0 = input0 - 0.1 * 2.1",
                    inputVectorWidth=1,
                    outputVectorWidth=1,

                    action="create"
                  ];

O2Calibrate::in   [ name="in",
                    rate=200.0,
                    action="assign" ];

O2Calibrate::out  [ name="out",
                    rate=200.0,
                    action="assign" ];

VOCCalibrate      [ dpe=DataPathArithmetic,
                    name="VOC Calibration",
                    device="computation",

                    initializer="",
                    arithmetic="output0 = input0 - 0.1 * 2.1",
                    inputVectorWidth=1,
                    outputVectorWidth=1,

                    action="create"
                  ];

VOCCalibrate::in        [ name="in",
                    rate=200.0,
                    action="assign" ];

VOCCalibrate::out [ name="out",
                    rate=200.0,
```

```
                action="assign" ];

COCalibrate     [ dpe=DataPathArithmetic,
                  name="CO Calibration",
                  device="computation",

                  initializer="",
                  arithmetic="output0 = input0 - 0.1 * 2.1",
                  inputVectorWidth=1,
                  outputVectorWidth=1,

                  action="create"
                ];

COCalibrate::in [ name="in",
                  rate=200.0,
                  action="assign" ];

COCalibrate::out [ name="out",
                  rate=200.0,
                  action="assign" ];

CO2Calibrate    [ dpe=DataPathArithmetic,
                  name="CO2 Calibration",
                  device="computation",

                  initializer="",
                  arithmetic="output0 = input0 - 30.1 * 0.5",
                  inputVectorWidth=1,
                  outputVectorWidth=1,

                  action="create"
                ];

CO2Calibrate::in        [ name="in",
                          rate=200.0,
                          action="assign" ];

CO2Calibrate::out [ name="out",
                          rate=200.0,
                          action="assign" ];

H2Calibrate     [ dpe=DataPathArithmetic,
                  name="H2 Calibration",
                  device="computation",

                  initializer="",
                  arithmetic="output0 = input0 - 30.1 * 0.5",
                  inputVectorWidth=1,
                  outputVectorWidth=1,
```

```
                    action="create"
                  ];

H2Calibrate::in   [ name="in",
                    rate=200.0,
                    action="assign" ];

H2Calibrate::out  [ name="out",
                    rate=200.0,
                    action="assign" ];

TempCalibrate     [ dpe=DataPathArithmetic,
                    name="Temp Calibration",
                    device="computation",

                    initializer="",
                    arithmetic="output0 = input0 - 30.1 * 0.5",
                    inputVectorWidth=1,
                    outputVectorWidth=1,

                    action="create"
                  ];

TempCalibrate::in         [ name="in",
                            rate=200.0,
                            action="assign" ];

TempCalibrate::out  [ name="out",
                    rate=200.0,
                    action="assign" ];

CPUTempCalibrate          [ dpe=DataPathArithmetic,
                    name="CPU Temp Calibration",
                    device="computation",

                    initializer="",
                    arithmetic="output0 = input0 - 3.1",
                    inputVectorWidth=1,
                    outputVectorWidth=1,

                    action="create"
                  ];

CPUTempCalibrate::in      [ name="in",
                            rate=200.0,
                            action="assign" ];

CPUTempCalibrate::out  [ name="out",
                    rate=200.0,
                    action="assign" ];
```

```
PressCalibrate   [ dpe=DataPathArithmetic,
                  name="Press Calibration",
                  device="computation",

                  initializer="",
                  arithmetic="output0 = input0 - 30.1 * 0.5",
                  inputVectorWidth=1,
                  outputVectorWidth=1,

                  action="create"
                ];

PressCalibrate::in       [ name="in",
                           rate=200.0,
                           action="assign" ];

PressCalibrate::out [ name="out",
                           rate=200.0,
                           action="assign" ];

HumidCalibrate   [ dpe=DataPathArithmetic,
                  name="Humid Calibration",
                  device="computation",

                  initializer="",
                  arithmetic="output0 = input0 - 30.1 * 0.5",
                  inputVectorWidth=1,
                  outputVectorWidth=1,

                  action="create"
                ];

HumidCalibrate::in       [ name="in",
                           rate=200.0,
                           action="assign" ];

HumidCalibrate::out [ name="out",
                           rate=200.0,
                           action="assign" ];

BatteryCalibrate         [ dpe=DataPathArithmetic,
                           name="Battery Calibration",
                           device="computation",
                           initializer="",
                           arithmetic="output0 = input0 / 32.0",
                           inputVectorWidth=1,
                           outputVectorWidth=1,

                           action="create"
                ];
```

```
BatteryCalibrate::in     [ name="in",
                           rate=200.0,
                           action="assign" ];

BatteryCalibrate::out    [ name="out",
                           rate=200.0,
                           action="assign" ];

#
# Connect the Calibration transforms to ADC outputs.
#

adc::O2out        -> O2Calibrate::in
  [ name="connection", type="physical", label="conn", action="create" ];

adc::CO2out       -> CO2Calibrate::in
  [ name="connection", type="physical", label="conn", action="create" ];

adc::COout        -> COCalibrate::in
  [ name="connection", type="physical", label="conn", action="create" ];

adc::VOCout       -> VOCCalibrate::in
  [ name="connection", type="physical", label="conn", action="create" ];

adc::PressOut     -> PressCalibrate::in
  [ name="connection", type="physical", label="conn", action="create" ];

adc::HumidOut     -> HumidCalibrate::in
  [ name="connection", type="physical", label="conn", action="create" ];

adc::TempOut      -> TempCalibrate::in
  [ name="connection", type="physical", label="conn", action="create" ];

adc::H2out        -> H2Calibrate::in
  [ name="connection", type="physical", label="conn", action="create" ];

adc::BatteryOut -> BatteryCalibrate::in
  [ name="connection", type="physical", label="conn", action="create" ];

#
# Create a DataPathFile to store all of this data.
#

outputFile        [ dpe=DataPathFile,
                    name="output file",
                    device="datapathfile",
#                   directory="/tmp/AQUATSstore",
                    directory="$AQUATSFILE",
                    access="w",
                    action="create" ];
```

```
#
# Obtain an alias for the output file's wildcard sink.
#

outputFile::wildSink [ name="*",
                       direction="sink",
                       wildcard=1 ];


#
# Attempt a connection between signals process from the input file and the
# output.
#

adc::O2out -> outputFile::wildSink
        [ name="raw O2 volts", type="physical", action="create" ];

O2Calibrate::out -> outputFile::wildSink
        [name="O2", type="physical", action="create" ];

adc::CO2out -> outputFile::wildSink
        [ name="raw CO2 volts", type="physical", action="create" ];

CO2Calibrate::out -> outputFile::wildSink
        [name="CO2", type="physical", action="create" ];

adc::COout -> outputFile::wildSink
        [ name="raw CO volts", type="physical", action="create" ];

COCalibrate::out -> outputFile::wildSink
        [name="CO", type="physical", action="create" ];

adc::VOCout -> outputFile::wildSink
        [ name="raw VOC volts", type="physical", action="create" ];

VOCCalibrate::out -> outputFile::wildSink
        [name="VOC", type="physical", action="create" ];

adc::PressOut -> outputFile::wildSink
        [ name="raw Pressure volts", type="physical", action="create" ];

PressCalibrate::out -> outputFile::wildSink
        [name="Pressure", type="physical", action="create" ];

adc::HumidOut -> outputFile::wildSink
        [ name="raw Humidity volts", type="physical", action="create" ];

HumidCalibrate::out -> outputFile::wildSink
        [name="Humidity", type="physical", action="create" ];

adc::TempOut -> outputFile::wildSink
```

```
        [ name="raw Temperature volts", type="physical", action="create" ];

TempCalibrate::out -> outputFile::wildSink
        [name="Temperature", type="physical", action="create" ];

adc::BatteryOut -> outputFile::wildSink
        [ name="raw Battery volts", type="physical", action="create" ];

adc::CPUTempOut -> outputFile::wildSink
        [ name="raw CPU temp volts", type="physical", action="create" ];

CPUTempCalibrate::out -> outputFile::wildSink
        [name="CPU Temperature", type="physical", action="create" ];

adc::O2status -> outputFile::wildSink
        [ name="raw O2 status volts", type="physical", action="create" ];

BatteryCalibrate::out -> outputFile::wildSink
        [ name="battery remaining", type="physical", action="create" ];

#
# Create the display.
#

ui              [ dpe=WaveUI,
                  name="user interface",
                  control="$AQCONTROLFILE",
                  width="800",
                  height="600",
                  trace.0="CO",
                  trace.1="CO2",
                  trace.2="VOC",
                  trace.3="O2",
                  trace.4="H2",
                  trace.5="temp",
                  trace.6="humid",
                  trace.7="press",
                  action="create" ];

ui::COin        [ name="CO 0",
                  rate=200.0,
                  action="assign" ];

ui::CO2in       [ name="CO2 1",
                  rate=200.0,
                  action="assign" ];

ui::VOCin       [ name="VOC 2",
                  rate=200.0,
                  action="assign" ];
```

```
ui::O2in          [ name="O2 3",
                    rate=200.0,
                    action="assign" ];

ui::H2in          [ name="H2 4",
                    rate=200.0,
                    action="assign" ];

ui::Tempin        [ name="temp 5",
                    rate=200.0,
                    action="assign" ];

ui::Humidin       [ name="humid 6",
                    rate=200.0,
                    action="assign" ];

ui::Pressin       [ name="press 7",
                    rate=200.0,
                    action="assign" ];

COCalibrate::out -> ui::COin
        [name="CO", type="physical", action="create" ];

CO2Calibrate::out -> ui::CO2in
        [name="CO2", type="physical", action="create" ];

VOCCalibrate::out -> ui::VOCin
        [name="VOC", type="physical", action="create" ];

O2Calibrate::out -> ui::O2in
        [name="O2", type="physical", action="create" ];

H2Calibrate::out -> ui::H2in
        [name="H2", type="physical", action="create" ];

TempCalibrate::out -> ui::Tempin
        [name="temp", type="physical", action="create" ];

PressCalibrate::out -> ui::Humidin
        [name="press", type="physical", action="create" ];

HumidCalibrate::out -> ui::Pressin
        [name="humid", type="physical", action="create" ];
```

References

1. (2004). Contract W7711-036681/001/TOR, Software Maintenance and Support.

2. (2004). OpenDX: Open Visualization Data Explorer (Online). OpenDX Organization. http://www.opendx.org (March 10, 2004).

3. (2004). NCSA HDF Home Page (Online). National Center for Supercomputing Applications. http://hdf.ncsa.uiuc.edu (March 10, 2004).

4. (2000). dicom2 (Online). Sebastian Barre. www.barre.nom.fr/medical/dicom2 (March 10, 2004).

5. (1997). MIR DICOM Central Test Node (Online). Electronic Radiology Laboratory, Mallinckrodt Institute of Radiology. http://wuerlim.wustl.edu/DICOM/ctn.html (March 10, 2004).

6. Tom Gee, William Fraser (2003). Software architecture for modeling and implementing dynamic, heterogenous acquisition and control networks. Technical Report. DRDC Toronto.

7. (2001). Contract W7711-017733/001/TOR, Design of System Architecture for Multi-Station Battlefield Biomedical Signal Acquisition System.

8. Tom Gee, Edward Kozak, Frank Ing, William Fraser (2003). A database schema for storing varied data types and configurations associated with physiological and environmental monitoring and control. Technical Report. DRDC Toronto.

9. (2004). The PDP++ Software Home Page (Online). Center for Neural Basis of Cognition, Carnegie Mellon University. http://www.cnbc.cmu.edu/Resources/PDP++/PDP++.html (July 4, 2004).

10. (2003). PDP++ Software Users Manual (Online). Chandley K. Dawson, Randall C. O'Reilly, James L. McClelland. file:///usr/local/pdp++/manual/html/pdp-user-1.html (July 20, 2004).

11. Tom Gee, William Fraser (2003). Prototype implementation of a medical monitoring and control system based on GUARDIAN architecture. Technical Report. DRDC Toronto.

12. (2003). graphviz - open source graph drawing software (Online). AT&T. http://www.research.att.com/sw/tools/graphviz (July 28, 2004).

13. (2004). PostgreSQL: The most advanced Open Source database system in the world (Online). PostgreSQL Global Development Group. http://www.postgresql.org (July 20, 2004).

14. (2003). Konqueror - Web Browser, File Manager - and more! (Online). KDE e. V.. http://konqueror.kde.org (July 20, 2004).

15. (2004). the KDE Desktop Environment (Online). KDE e. V.. http://www.kde.org (July 20, 2004).

16. (2001). The Omnivore: KDE's flexible I/O architecture (Online). c't magazine. http://www.heise.de/ct/english/01/05/242/ (March 10, 2004).

17. (2004). Qt Overview (Online). Trolltech Inc.. http://www.trolltech.com/products/qt/index.html (July 20, 2004).

18. Free Software Foundation Inc. (1991). GNU General Public License. Version 2.

19. (2001). GnomeVFS - Filesystem Abstraction Library (Online). Seth Nickel. http://developer.gnome.org/API/gnome-vfs (March 10, 2004).

20. (2004). AVFS - A Virtual File System (Online). Miklos Szeredi. http://sourceforge.net/projects/avf (July 20, 2004).

21. Free Software Foundation Inc. (1991). GNU Library General Public License. Version 2.

22. (2003). Contract W7711-037867/A, Enhancement of integrated computer architecture sub-system components for monitoring armoured vehicles and submarine air quality.

23. R. Greg Lavender, Douglas C. Schmidt (1995). Active Object - an Object Behavioral Pattern for Concurrent Programming. In *Proceedings of the Second Pattern Languages of Programs Conference*.

24. (2000). ParaGUI Cross-Platform Widget Set (Online). ParaGUI Project, Alexander Pipelka. http://www.paragui.org (May 7, 2004).

25. (2003). Simple MediaDirect Layer (SDL) (Online). Sam Latinga. http://www.libsdl.org (October 1, 2003).

26. (2003). Open Graphics Library OpenGL (Online). OpenGL. http://www.opengl.org (October 1, 2003).

27. Schmidt, Douglas C. (1993). The ADAPTIVE Communication Environment: An Object-Oriented Network Programming Toolkit for Developing Communication Software. In *12th Sun User Group Conference*.

Index

ACE, 13
aedGUI, 13

C++, 2, 14
configuration editor, iii, v, 6, 14, 21

database, iii, v, 1, 8, 21
datatrac, 1, 8, 14
DICOM, 1, 2, 4
dicom2, 4
dot, 6

EEG, 1, 10
EMG, 10

filesystem, iii, 8, 14, 21
flows
    sink, 6
    source, 6
FUSE, 9

GLGraph, 12
gnomeVFS, 9
GPL, 8, 9
GraphTrace, 12
graphviz, 6
GUARDIAN, iii, v, 1, 6, 8, 10, 13, 21

HDF5, 1, 2, 14

IMEOS, 1

KDE, 8
KIO, 8
konqueror, 8

LGPL, 9
Linux, 1, 13

Microsoft Windows, 9
modeling, iii, v, 21
MRI, 4

nautilus, 9

OpenDX, 1, 2, 4, 14
OpenGL, 12, 13

paraGUI, 12, 13
PDP++, 1, 2, 14
PostgreSQL, 6
processing network, 6

Qt, 8

SDL, 12, 13

TraceGraphElement, 13

user interface, 1, 10

WaveUI, 12

DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when document is classified)

1. ORIGINATOR (the name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Centre sponsoring a contractor's report, or tasking agency, are entered in section 8.)

Covenant Systems Development
27 Bray St, Beeton, Ontario, L0G 1A0

2. SECURITY CLASSIFICATION (overall security classification of the document including special warning terms if applicable).

UNCLASSIFIED

3. TITLE (the complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S,C,R or U) in parentheses after the title).

Software Maintenance and Support Contract — Final Report

4. AUTHORS (Last name, first name, middle initial. If military, show rank, e.g. Doe, Maj. John E.)

Gee, Tom

5. DATE OF PUBLICATION (month and year of publication of document)	6a. NO. OF PAGES (total containing information. Include Annexes, Appendices, etc).	6b. NO. OF REFS (total cited in document)
September 2004	107	27

7. DESCRIPTIVE NOTES (the category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered).

Contractor Report

8. SPONSORING ACTIVITY (the name of the department project office or laboratory sponsoring the research and development. Include address).

Defence Research & Development - Toronto
PO Box 2000, 1133 Sheppard Ave. West, Toronto, Ontario, CANADA M3M 3B9

9a. PROJECT OR GRANT NO. (if appropriate, the applicable research and development project or grant number under which the document was written. Specify whether project or grant).	9b. CONTRACT NO. (if appropriate, the applicable number under which the document was written).
	W7711-036681/001/TOR

10a. ORIGINATOR'S DOCUMENT NUMBER (the official document number by which the document is identified by the originating activity. This number must be unique.)	10b. OTHER DOCUMENT NOs. (Any other numbers which may be assigned this document either by the originator or by the sponsor.)

11. DOCUMENT AVAILABILITY (any limitations on further dissemination of the document, other than those imposed by security classification)

(X) Unlimited distribution
( ) Defence departments and defence contractors; further distribution only as approved
( ) Defence departments and Canadian defence contractors; further distribution only as approved
( ) Government departments and agencies; further distribution only as approved
( ) Defence departments; further distribution only as approved
( ) Other (please specify):

12. DOCUMENT ANNOUNCEMENT (any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution beyond the audience specified in (11) is possible, a wider announcement audience may be selected).

13. ABSTRACT (a brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), (R), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual).

To provide an effective platform for acquiring and processing a large number of high-speed channels, we were engaged to explore several third party software packages. We were also tasked with augmenting the existing software architecture, the GUARDIAN system.

We examined packages for real-time modeling, display, storage and importing outside data. We evaluated the tasks required to configure and apply GUARDIAN to this type of problem, as well as those required to navigate the stored data after acquiring. For those tasks which were onerous, we explored ways of simplifying them. We examined various options for creating a graphical configuration editor, a filesystem-type interface to the storage database, and a real-time waveform-oriented user interface, with the latter proceeding through the design and initial implementation phases.

The existing system, combined with the third party packages, is a suitable platform for this research, if somewhat awkward in its employ. By completing the modest projects started under this program, the ease and efficiency of the system will be increased markedly.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus. e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus-identified. If it not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title).